Universidad Autónoma Metropolitana

127.

# TMS32010

C.B.I.

## SIMULADOR

Ing. Computación

Ma. del Pilar Villa Cervantes
Alberto Garcia Castillo
Diciembre 1992

**Proyecto Terminal :** Simulador TMS32010

**Asesor de Proyecto :** MC. Oscar Yañez Suárez
Coordinador de Ingenieria Biomedica

Diciembre de 1992

# Análisis e Implemetación del TMS32010

**Objetivo :** Analizar la Aquitectura, flujo de datos y caracteristicas de los componentes del micro-procesador digital de señales TMS32010 para así poder implementación en Lenguaje C las estructuras de datos que serviran para programar un simulador que muestre el proceso de programas codificados (ensamblador) de las instrucciones que lo componen.

Una vez realizado el análisis e implementación se programara como parte 2 de éste proyecto el simulador. El objetivo principal del proyecto es el de poder proporcionar una herramienta de laboratorio para poder comprobar el funcionamiento del Chip en alguna aplicación real en donde esté involucrado.

Así mismo se realizará una clasificación del conjunto de instrucciones para su mejor manejo al momento de realizar la decodificación de código.

**Introducción :** El TMS32010 pertenece a una familia de procesadores digitales de 16/32 bits muy poderosa debido a que cuenta con un número grande de instrucciones muy eficientes y faciles de programar pudiendo ejecutar hasta 5 millones de instrucciones por segundo.

Sus aplicaciones son muchas y es por ello de la necesidad de un análisis detallado el cual lograra que el programador cuente con una definición veridica y sencilla de las partes que componen a dicho procesador.

La entrada estará compuesto por código en ensamblador y el cual sera buscado en una serie de tablas definidas y clasificadas de acuerdo al parentesco que tengan en la parte más significativa de los 16 bits que lo componen. De igual forma iremos analizando las partes que componen la arquitectura del TMS32010 para que segun sea la necesidad y aplicación del componente se pueda definir una estructura de datos sencilla pero a la vez eficiente, sin embargo, no sera propósito de éste proyecto la implementación y programación de las macro-instrucciones con las que cuenta el procesador debido a que se perderia la linealidad de la secuencia de registros y demas componentes que van cambiando en el transcurso del proceso de datos.

Una ventaja del conjunto de instrucciones con las que cuenta el TMS32010 es el de que gracias a su longitud de palabra la ejecución de una instrucción sera realizada en mucho menos pasos que si fuera de 8 bits.

# Arquitectura y Análisis de Instrucciones

El TMS32010 está basado en una arquitectura de Acumulador el cual tiene como función el servir como medio de almacenamiento para datos y resultados de operaciones de la ALU a la vez que sirve como alimentador en muchos casos de las datos que requiere la misma ALU. Cuenta con caracteristicas muy singulares y sorprendentes pero en realidad solo 4 de ella lo distinguen de otros procesadores digitales :

**1.-** 16 bits el longitud de palabra y acumulador de 32 bits.

**2.-** Multiplicador rápido de ciclo sencillo de 200 ns. y desplazador de más de un bit para corrimiento de datos dentro de la ALU.

**3.-** Ciclo de instrucciones rápido (casi todas las instrucciones a 200 ns. en un ciclo).

**4.-** 144 registros de proposito general (Texas Instrument los llama memoria de datos).

Este procesador utiliza Arquitectura Harvard en la cual la memoria de datos y la de programas permanece en 2 espacios separados . Esto permite un traslapamiento por completo del Fetch de la instrucción (el cual no se contemplara en el simulador) y su ejecucón.

## Componentes de la Arquitectura

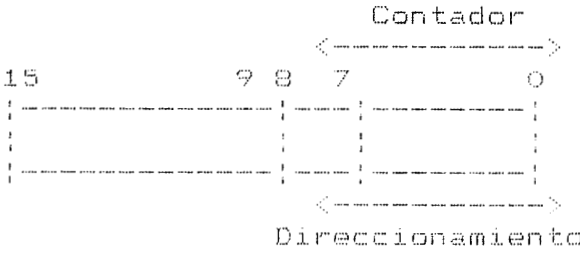
**Unidad** : Acumulador
**Símbolo** : ACC
**Función** : Acumulador de 32 bits.
Almacena la salida de la ALU y frecuentemente es tambien la entrada de la ALU. Opera con 32 bits y se compone de la parte alta (bits 31 a 16) y parte baja (bits 15 a 0). Existen instrucciones para guardar las 2 partes, cada una en la memoria.

**Implementación** : int ACC[2]

**Unidad** : Unidad Aritmetica Lógica
**Símbolo** : ALU
**Función** : Unidad Lógica y Aritmética de 2 puertos de 32 bits de propósito general que opera con datos de 32 bits. Esta unidad suma,resta y ejecuta operaciones lógicas . El acumulador es siempre el destino y el operador primario.

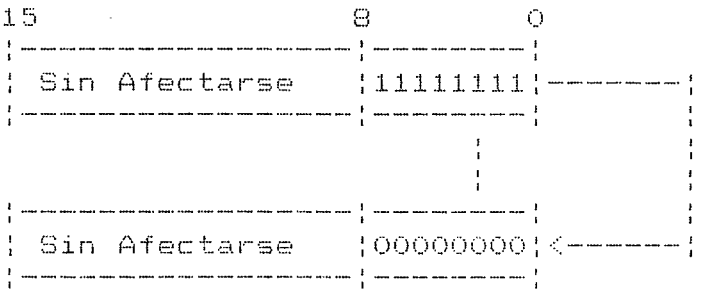
**Implementación** :
```
Función ALU(operando,operador1,operador2)
        caso operando
             caso Suma  :(operador1+operador2)
             caso Resta :(operador1-operador2)
             caso Mult. :(operador1*operador2)
             caso Div.  :(operador1/operador2)
             caso And   :(oper1 .and. oper2)
             caso Or    :(oper1  .or. oper2)
             caso Not   :(~oper1)
        fin-caso
```

**Unidad** : Registros Auxiliares
**Símbolo** : AR0,AR1
**Función** : 2 Registros de 16 bits cada uno para direccionamiento indirecto de la memoria de datos y control de la cuenta de ciclo. Los 9 bits menos significativos de cada registro son configurados como contadores bidireccionales, y en general para almacenar datos temporales. Cuando el Registo Auxiliar se autoincrementa /decrementa por una instrucción de direccionamiento indirecto o por BANZ, los 9 bits menos significativos son afectados, uno más que los 8 bits de la parte baja es usada para el direccionamiento indirecto.

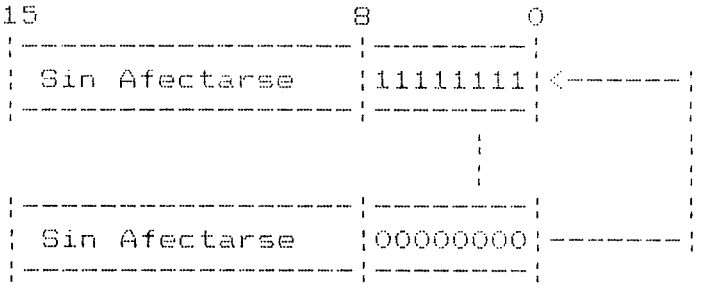


Esta posición del contador de un registro auxiliar es un contador auxiliar.

Autoincremento




Atodecremento




**Implementación :** int unsigned AR[2];

**Unidad :** Registro de Estados
**Símbolo :** **DP** - Apuntador a la página de memoria de datos
**ARP** - Contador al registro auxiliar en uso
**INTF**- Registro bandera de un solo bit para indicar el requerimiento de una interrupción
**INTM**- Registro de modo de un solo bit que enmascara la bandera de interrupción
**OV** - Registro bandera de un solo que indica un sobreflujo en operaciones aritmeticas
**OVM** - REgistro de modo de un solo bit que define modo de saturación / insaturación de operaciones aritmeticas
**Función :** **OV** - Registro Bandera de Sobreflujo del Acumulador

0 --> El Acumulador no tiene sobreflujo
1 --> Ha ocurrido un sobreflujo en el acumulador

La instrucción BV (salta si existe sobreflujo) puede aclarar éste bit y causar un salto si éste es seteado.

**OVM** - Bit de modo de sobreflujo

0 --> Deshabilita el modo de sobreflujo
1 --> Habilita el modo de sobreflujo

La instrucción SOVM carga el bit OVM con 1; la instrucción ROVM carga el bit OVM con 0.

**INTM** - Bit enmascarable de la interrupción

0 --> Selecciona el registro AR0
1 --> Selecciona el registro AR1

El ARP sólo puede ser modificada ejecutando las instrucciones MAR ó LARP ó por instrucciones que permitan la opción de direccionamiento indirecto.

**DP** - Apuntador a la página de la memoria de datos

0 --> Selecciona las primeras 128 palabras de la memoria de datos
1 --> Selecciona las ultimas 16 palabras de la memoria de datos, es decir, la página 1

El bit DP tambien puede ser modificado ejecutando las instrucciones LDP ó LDPK.

Todos los registros de estados se almacenan en una misma palabre de estado a traves de la instrucción SST usando modo de direccionamiento directo y sólo puedde ser especificado una dirección menor a 16 palabras, debido a que a que la 2a página de memoria contiene solo 16 palabras . Si se seleciona el modo de direccionamiento indirecto, el contenido del registro de status puede ser almacenado en cualquier localidad de la RAM despues de ejecutarse la instrucción SST:

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| OV | OVM | INTM | 1 | 1 | 1 | 1 | ARP | 1 | 1 | 1 | 1 | 1 | 1 | /// | DP |

**Implementación** :
```
#define DP    0x1
#define ARP   0x100
#define INTM  0x1000
#define OVM   0x2000
#define OV    0x4000
Unsigned int STATUS (Operaciones c/word)
```

**Unidad** : Ram de Datos
**Símbolo** : RAM Memoria de Acceso Aleatorio
**Función** : La memoria de Datos consiste en 144 palabras de 16 bits de ancho presente dentro del chip. Todos los operandos que no son inmediatos residen dentro de ésta RAM. Así la memoria de datos se comporta como los registros de proposito general de otras computadoras y se tiene la facilidad de transferir datos a la memoria de programas, aunque el proceso es lento en el TMS32010 más sin embargo en las aplicaciones reales esto es de gran ayuda.

**Implementación** : int RAM[144]

**Unidad** : Pila de Direcciones
**Símbolo** : PILA
**Función** : Almacena las direcciones del contador de programa en una llamada a subrutina o de una interrupción y la cual consta de 4 registros de de 12 bits. La instrucción PUSH coloca los 12 bits menos significativos del acumulador en el tope de la Pila, mientras la instrución POP coloca el tope de la Pila en los 12 bits menos significativos del acumulador. Siguiendo la instrucción POP el tope de la pila puede ser movido dentro de la memoria de datos por almacenamiento de la parte baja del acumulador. Estp permite la expansión de la Pila dentro de la RAM de datos.

**Implementación** :
```
Struct PILA {
    int DIR[4];
    int Cont;
};
```

**Unidad** : Contador de Programas
**Símbolo** : PC
**Función** : Registro de 12 bits y que contiene la dirección de memoria de programa siempre de la siguiente instrucción a ser ejecutada. Este dispositivo lee la instrucción de la localidad de la memoria de programa direccionada por el PC e incrementa el PC preparandola para la siguiente instrucción de fetch.

**Implementación** ; int PC

**Unidad** : Registro P (Producto)
**Símbolo** : P
**Función** : Registro de 32 bits que contiene el producto de las operaciones de multiplicación

**Implementación** : Long int P

**Unidad** : Registro T (Producto)
**Símbolo** : T
**Función** : Registro de 16 bits que contiene el multiplicando durante la operación de multiplicación

**Implementación** : int T

**Unidad** :Memoria de solo lrctura de Programas
**Símbolo** :ROM
**Función** :Almacenamiento del código del programa fuente que tiene una longitud de 1536 palabras de 16 bits y junto con el PC van indicando que instrucción se va ejecutando el programa

**Implementación** :int ROM[1536]

## Principales Procesos de Operación

### Multiplicador

El multiplicador de 16 x 16 bits consta de los registros T, P y el arreglo para la multiplicación. El registro T es de 16 bits y almacena el multiplicando, mientras que el registro P es de 32 bits y almacena el producto. El procedimiento para usar el multiplicador es : el multiplicando debe ser cargado en el registro T desde la RAM de datos usando alguna de las siguientes instrucciones LT, LTA, o LTD, entonces se ejecuta la instrucción MPY( Multiplicación) o MPYK (Multiplicación Inmediata) . Si se usa MPY el valor del multiplicador es un número de 16 bits en la RAM de datos y si se uba MPYK el valor del multiplicador es una constante inmediata de 13 bits derivada de la palabra de la instrucción MPYK, ésta constante de 13 bits está justificada a la derecha y con signo extendido. Al final de la multiplicación el resultado se encontrara en el registro P. El resultado podra ser cargado o sumado dentro del acumulador ejecutando alguna de las siguientes instrucciones APAC, SPAC, LTA, LTD o PAC.

### Desplazadores

Existen 2 tipos de desplazadores para el manejo de manipulación de datos : Un desplazador con más de un bit (Barrel Shifter) para desplazamientos de datos desde la RAM, dentro de la ALU y un desplazador Paralelo para desplazamientos del acumulador dentro de la RAM de Datos

## Desplazador de más de un bit (Barrel Shifter)

Este desplazador ejecuta corrimientos a la izquierda de 0 a 15 lugares en todas las palabras de la memoria de datos que son cargadas en, restadas de o sumadas al acumulador y que para fines de opreaciones seran representadas con operaciones de multiplicaciones de valores del tipo 2 EXP N, donde N petenece a 0,1,2,3,4,5,6,7,8,9.... 15. Así mismo éste desplazador llena de ceros los bits de la parte baja y extiende el signo de la palabra de 16 bits de la memoria de datos a 32 bits por lo que es llamado un corrimiento aritmético a la izquierda.

## Desplazador Paralelo

El desplazador paralelo es activado solo por la instrucción almacena la parte alta del acumulador SACH. Este desplazador ejecuta un corrimiento a la izquierda del total de los 32 bits del acumulador y coloca 16 bits dentro de la RAM de datos, resultando uns pérdida en los bits de la parte alta del acumulador. Así solo podra hacer desplazamientos de solo 0, 1 o 4 bits. Corrimientos de 1 o 4 bits seran ejecutados cuando se elijan operaciones de multiplicación. Los corrimientos a la derecha no son implementados directamente.

## Direccionamientos a la Memoria de Datos

Existen 3 modos de direccionamiento : Directo, Indirecto e Inmediato .

- ***Direccionamiento Indirecto :***

El direccionamiento Indirecto utiliza los 8 bits de la parte baja de los registros auxiliares como la dirección a la memoria de datos, esto es suficiente para direcionar todas las 144 palabras de datos , no es necesario paginar con direccionamientos Indirectos . Así los registros auxiliares pueden autoincrementarse /decrementarse durante cualquier instrucción indirecta. El incremento/decremento ocurre despues de que la instrucción ha sido terminada de ejecutarse.

Dentro del código de las instrucciones se tiene un bit dentro de la palabra a decodificar durante la ejecución del programa el cual nos indica el modo de direccionamiento y con ayuda de los registros auxiliares AR0 y AR1 obtendremos la dirección deseada de los datos requeridos para obtener los datos de la memoria.

ARP < AR0 / AR1

- ***Direccionamiento Directo :***

En el direccionamiento directo, siete bits de la palabra de instrucción son concatenados con el apuntador a la página de datos (DP) para formar la dirección a la memoria de datos. Este direccionamiento utiliza el siguiente esquema :

| DP | LOCALIDAD DE MEMORIA |
|---|---|
| 0 | 0 -- 127 |
| 1 | 128 -- 144 |

DP es parte del registro de Status y de éste modo puede ser almacenado en la memoria de datos.

DP + DMA = Dirección de los datos en la Memoria

- ***Direccionamiento Inmediato :***

El conjunto de instrucciones del TMS32010 contiene instrucciones "Inmediatas" especiales, tales como MPYK, LACK y LARK . Estas instrucciones derivan datos de parte de la palabra de instrucción mejor que la RAM de datos. Es decir, en la misma palabra viene implicita la dirección del dato a ser operado.

Algunas Estrategias a seguir

Cuando la Instrucción se ejecuta realiza uno o varios ciclos, estos ciclos consisten principalmente de 2 pasos :

```
                ¦  i) Fetch
        Ciclo <
                ¦ ii) Decodifica la Instrucción
```

para su ejecución posterior.

La Idea General del sistema es el de manejar todo atraves de una tabla que nos resualva el problema de la ejecución de la instrucción :

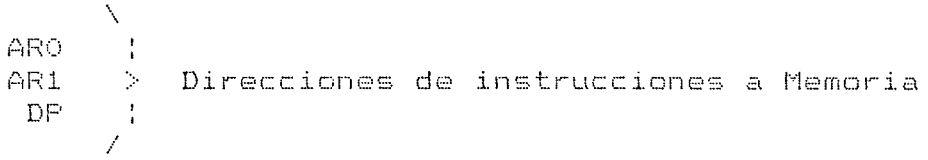
| Nemónicos | Códigos | Función | Tamaño |
|---|---|---|---|
| A | 1111001101000011 | ACC <--AR1 | 16 bits |
| ADD A,* | 0001101111101111 | ACC <--ACC+ARO* | 32 bits |
| | | | |
| | | | |

La idea de la decodificación es clasificar las instrucciones que comiencen con un número determinado de bits iguales y unicamente decodificar (analizar) el resto de los bits de la palabra en cuentión.

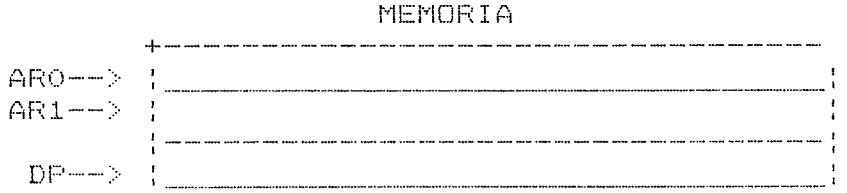
```
                                  +------Esta parte siempre varia
                                  ¦      deacuerdo al prpgramador
                                  V
              ---------------------------------------
      +--->¦  OPCODE     ¦  OPERANDO              ¦
      ¦    ---------------------------------------
      ¦
      ¦
  Es fijo para las Instrucciones
```

Clasificación de Registros y Banderas

```
        \
  ACC    ¦
   D     >    Operaciones Aritmetico Lógicas
   T     ¦
        /
```




La memoria será dividida en 3 partes para poder controlar la memoria que le corresponde a ARO, AR1, y a DP y esto tambien facilitara la búsqueda de las direcciones al momento de utilizar a cualquiera de éstos registros de direcciones.



## *FLUJO DE DATOS*

*1.-* Leer del archivo la dirección de la primera instrucción y asignarla al PC.

*2.-* Leer la instruccion

*3.-* Realizar una búsqueda de la instrucción en una tabla para determinar a que instrucción pertenece.: esta búsqueda se hará en base a los primeros 4 bits puesto que estos estan fijos; en el caso de haber instrucciones que tengan los mismos 4 bits , se buscará con los siguientes 4 bits y así sucesivamente.

| 4 BITS | INSTRUCCION | CODIGO FIJO (hex.) |
|---|---|---|
| 0000 | ADD | 0 |
| 0001 | SUB | 1 |
| 0010 | LAC | 2 |
| 0011 | LAR | 3 |
| 0011 | SAR | 3 |
| 0100 | IN | 4 |
| 0100 | OUT | 4 |
| 0101 | SACH | 5 |
| 0101 | SACL | 5 |
| 0110 | ADDH | 60 |
| 0110 | ADDS | 61 |
| 0110 | SUBS | 64 |
| 0110 | SUBH | 62 |
| 0110 | SUBS | 63 |
| 0110 | ZALH | 65 |
| 0110 | ZALS | 66 |
| 0110 | LARP | 68 |
| 0110 | MAR | 68 |
| 0110 | LDP | 6F |
| 0110 | LDPK | 6E0 |
| 0110 | LT | 6A |
| 0110 | LTA | 6C |
| 0110 | LTD | 6B |
| 0110 | MPY | 6D |
| 0110 | DMOV | 69 |
| 0110 | TBLR | 67 |
| 0111 | LARK | 7 |
| 0111 | AND | 79 |
| 0111 | LACK | 7E |
| 0111 | OR | 7A |
| 0111 | XOR | 78 |
| 0111 | LST | 7B |
| 0111 | SST | 7C |
| 0111 | TBLW | 7D |
| 0111 | ABS | 7F88 |
| 0111 | ZAC | 7F89 |
| 0111 | CALA | 7F8C |
| 0111 | RET | 7F8D |
| 0111 | APAC | 7F8F |
| 0111 | PAC | 7F8E |
| 0111 | NOP | 7F80 |

| | | |
|---|---|---|
| 0111 | DINT | 7F81 |
| 0111 | EINT | 7F82 |
| 0111 | ROVM | 7F8A |
| 0111 | SOVM | 7F8B |
| 0111 | SPAC | 7F90 |
| 0111 | POP | 7F9D |
| 0111 | PUSH | 7F9C |
| 100 | MPY | |
| 1111 | B | F200 |
| 1111 | BANZ | F400 |
| 1111 | BGEZ | FD00 |
| 1111 | BGZ | FC00 |
| 1111 | BIOZ | F300 |
| 1111 | BLEZ | FB00 |
| 1111 | BLZ | FA00 |
| 1111 | BNZ | FE00 |
| 1111 | BV | F500 |
| 1111 | BZ | FF00 |
| 1111 | CALL | F800 |

**4.-** Ya determinada a que instrucción pertenece, se llamará a la función que realizará dicha operación.

**5.-** desglosar la instrucción : Es decir determinar el significado de cada uno de los bits que la componen dependiendo al tipo de instrucción a la que pertenece.

### *- INSTRUCCIONES CON EL ACUMULADOR:*

En la mayoria de las instrucciones con el acumulador el codigo de instrucción se encuentra de la siguiente manera:

- Del bit 0 al bit 6 contiene la dirección de memoria del dato.
- El bit 7 nos indica el modo de direccionamiento, 0 cuando el direccionamiento es directo y 1 cuando es indirecto.
- En algunas instrucciones como ADD, LAC, y SUB los bits del 8 al 11 indica el corrimiento, el cual puede ser de 0 a 15. En la instrucción SACH el corrimiento puede ser de solamente 0, 1 ó 4.

*- INTRUCCIONES DE SALTO :*

El codigo de intrucción de salto consta de dos palabras de 16 bits; la primera palabra contiene el codigo de dicha instrucción y la segunda palabra de los bits del 12 al 15 tienen 0's y del bit 0 al 11 contiene la dirección hacia a donde va a brincar; excepto las intrucciones CALA, y RET tienen solo una palabra de 16 bits cuyos codigos son 7F8C y 7F8D. respectivamente.

*- REGISTRO P, T E INSTRUCCIONES DE MULTIPLICACION :*

Las siguientes instrucciones: LT, LTA, LTD; tienen en la palabra de instrucción, del bit 8 al 15 el código de instrucción , el bit 7 indica el modo de direccionamiento y del bit 0 al 6 la dirección del valor que se cargara en el registro T.

*- INSTRUCCIONES DE CONTROL :*

En la mayoria de las instrucciones de control su codigo de instrucción es fijo excepto LST y SST; los bits del 0 al 6 contienen una dirección de memoria.

*-INSTRUCCIONES DE ENTRADA Y SALIDA Y OPERACIONES CON LA MEMORIA DE DATOS*

En estas instrucciones del bit 0 al 6 se especificara la dirección de memoria . Solo en las istrcciones de INT y OUT del bit 8 al 10 se especifica el puerto.

*-INSTRUCCIONES CON REGISTRO AUXILIAR Y APUNTADOR A LA PAGINA DE DATOS*

En algunas instrucciones como LAR, LARK,Y SAR en el bit 8 se especifica el registro auxiliar usado, 0 para el registro R0 y 1 para el registro R1.

En las instrucciones LARP Y LDPK solo es especificado el último bit por la misma instrucción ; en el caso de LARP se especifica el registro auxiliar y en LDPK el apuntador de página.

6.- Una vez decodificada la instrucción y analizada, es ejecutada.

***ABS*** : Valor absoluto del acumulador.
***Código*** = 7F88.
***Descripción*** : obtiene el valor absoluto del acumulador.

***ADD*** : Suma al acumulador con desplazamiento.
***Código***: 0[shift]XX h.
***Descripción*** : el contenido de la dirección de memoria de datos es desplazado a la izquierda y sumado al acumulador; este desplazamiento es indicado en el código de instrucción del bit 8 al 11 y por lo que el corrimiento puede ser de 0 a 15 bits . Durante el corrimiento los bits de orden bajo son rellenados con cero y los de orden alto son bits de signo; es decir el signo es extendido. y el resultado es guardado en el acumulador.

***ADDH*** : Suma a la parte alta del acumulador.
***Código*** = 60XX.h
***Descripción*** : suma el contenido de la memoria de datos a la parte alta del acumulador

***ADDS*** : Suma la parte baja del acumulador con el signo suprimido.
***Código*** = 61XX h
***Descripción*** : Suma el contenido de la dirección de memoria de datos con el contenido del acumulador sin extender el bit de signo.

***AND*** :And con los bits de orden bajo del acumulador
***Código*** = 79XX h.
***Descripción*** : Se realiza la operación AND con la parte baja del acumulador y el contenido de la memoria de datos; y la parte alta del acumulador con cero y esto concatenado se guarda en el acumulador.

***APAC*** : Suma el registro P con el acumulador.
***Código*** = 7F8f h.
***Descripción*** : El contenido del registro P, resultado de la multiplicación es sumado con el contenido del acumulador y almacenado en al acumulador.

***B*** : Salto incondicional.
***Código*** = F900 0XXX h.
***Descripción*** : Salta a la localidad del programa especificado por la dirección de memoria de programa.

***BANZ*** : Salto si el registro auxiliar es diferente de cero
***Código*** = F400 0XXX h.
***Descripción*** : Si los nueve bits bajos del registro auxiliar no son iguales a cero entonces el registro auxiliar es decrementado y la dirección de memoria de programa es cargado en el PC. Si estos bits son iguales a cero entonces el PC es incrementado en 2 y AR es decrementado.

***BGEZ*** : Salto si el acumulador es mayor o igual que cero.
***Código*** = FD00 0XXX h.
***Descripción*** : Si el contenido del acumulador es mayor o igual a cero entonces salta a la localidad de memoria de programa espacificada de otro modo el PC es incrementado en 2.

***BGZ*** : Salta siel acumulador es mayor que cero.
***Código*** = FC00 0XXX h.
***Descripción*** : Si el contenido del acumulador es mayor que cero salta a la localidad de memoria de programa espacificada de otro modo el PC es incrementado en 2.

***BLEZ*** : Salta si el acumulador es menor o igual que cero.
***Código*** = FB00 0XXX h.
***Descripción*** : Si el contenido del acumulador es menor o igual que cero salta a la localidad de memoria de programa espacificada de otro modo el PC es incrementado en 2.

***BLZ*** : Salta si el acumulador es menor que cero.
***Código*** = FA00 0XXX h.
***Descripción*** : Si el contenido del acumulador es menor que cero salta a la localidad de memoria de programa espacificada de otro modo el PC es incrementado en 2.

***BNZ*** : Salta si el acumulador es diferente de cero.
***Código*** = FE00 0XXX h.
***Descripción*** : Si el contenido del acumulador no es cero salta a la localidad de memoria de programa espacificada de otro modo el PC es incrementado en 2.

***BN*** : Salta si hay overflow.
***Código*** = F500 0XXX h.
***Descripción*** : Si la bandera de ov es encendida entonces salta a la localidad de memoria de programa espacificada y la bandera de ov es limpiada de otro modo el PC es incrementado en 2.

***BZ*** : Salta si el acumulador es igual a cero.
***Código*** = FF00 0XXX h.
***Descripción*** : Si el contenido del acumulador es igual a cero salta a la localidad de memoria de programa espacificada, de otro modo el PC es incrementado en 2.

***CALA*** : LLamada indirecta a subrutina.
***Código*** = 7F8C h.
***Descripción*** : El contenido del PC es incrementado y puesto en el tope de la pila; y los 12 bits menos significativos del acumulador son cargados en el PC.

***CALL*** : LLamada directa a subrutina.
***Código*** = F800 0XXXh.
***Descripción*** : El contenido del Pc es increntado en 2 y puesto en el tope de la pila y la dirección de memoria de programa es cargada en el PC.

***DINT*** : Deshabilita interrupción.
***Código*** = 7F81
***Descripción*** : La bandera de INTM es puesta en 1 Lógico para deshabilitar interrupción.

***DMOV*** : Movimientos de datos en memoria.
***Código*** = 69XX h.
***Descripción*** : El contenido de la dirección de memoria de datos es copiada en la siguiente dirección de la memoria de datos. Esta instrucción es subsecuente de la instrucción LTD.

***EINT*** : Habilita interrupción.
***Código*** = 7F82 h.
***Descripción*** : La bandera de INTM es limpiada poniendola en 0 Lógico.

***LAC*** : Carga acumulador con desplazamiento.
***Código*** = 2XXX h.
***Descripción*** : El contenido de la dirección de memoria de datos es cargada al acumulador con un desplazamiento a la izquierda ; que va de 0 a 15 bits; este desplazamiento es indicado dentro del código de operación en los bits del 8 al 11. Durante el desplazamiento, los bits de orden bajo son rellenados con cero y los bits de orden alto son de signo extendido.

***LACK*** : Carga al acumulador con 8 bits constante.
***Código*** = 7EXX h.
***Descripción*** : Los 8 bits constantes son cargados al acumulador justificados a la derecha .Los siguentes 24 bits superiores del acumulador son ceros; es decir el signo es suprimido.

***LAR*** : Carga el registro auxiliar.
***Código*** = 3XXX h.
***Descripción*** : el contenido de la dirección de memoria de datos especificada es cargado en el registro auxiliar designado en los bits de código de instrucción del 8 al 10.

***LARK*** : Carga al registro auxiliar designado con una constante de 8 bits
***Código*** = 7XXX h.
***Descripción*** : Los 8 bits son cargados en el registro auxiliar designado justificados a la derecha y rellenado los siguientes bits con cero ; es decir el bit de signo es suprimido.

***LARP*** : Carga registro auxiliar apuntado inmediato.
***Código*** = 688X h.
***Descripción*** : carga un bit constante que identifica al registro auxiliar deseado. Esta instricción es subsecuente de la instrucción MAR.

*LDP* : Carga el apuntador a la página de la memoria de datos

***Código*** = 6FXX h.

***Descripción*** : El bit menos significativo de la dirección de memoria de datos es cargado en el apuntador a la paágina de memoria de datos (DP).

*LDPK* : Carga el apuntador inmediato a la página de datos.

***Código*** = 6E0X h.

***Descripción*** : El bit menos significativo es cargado en el apuntador de página de memoria de datos. Si DP = 0, apunta de la 0-127 palabra y si DP =1 apunta de la 128-143 palabra.

*LST* : Carga el registro de estado en la memoria de datos.

***Código*** = 7BXX h.

***Descripción*** : Almacena el contenido del registro de estado. Esta instrucción es usada para reestablecer el estado despues de interrupciones y llamadas a subrutinas.

*LT* : Carga el registro T .

***Código*** = 6AXX h.

***Descripción*** ; Carga al registro T con el contenido de la dirección de memoria.

*LTA* : Carga registro T y Acumula previo producto.

***Código*** = 6CXX h.

***Descripción*** : El contenido de la dirección de memoria especificada es cargada en el registro T. El contenido del registro P (previo producto de la operación multiplicación) es sumado con el acumulador y el resultado almacenado con el acumulador.

*LTD* : Carga registro T ,Acumula previo producto y mueve memoria de datos.

Código = 6BXX h.

*Descripción* : El registro T es cargado con el contenido de la dirección de memoria de datos. El contenido de P es sumado con el acumuldor y el contenido de la dirección de memoria de datos especificada es transferida a la siguiente dirección.

***MAR*** : Modifica el registro auxiliar.
***Código*** = 68XX h.
***Descripción*** : Si la instrucción utiliza el modo de direccionamiento indirecto, incrementa y decrementa el registro auxiliar y esto hace cambiar el puntero del registro auxiliar actual. Si utiliza el modo de direccionamiento directo realiza lo mismo que la instrucción LARP.

***MPY*** : Multiplica.
***Código*** = 6DXX h.
***Descripción*** : El contenido de la dirección de la memoria de datos es multiplicado con el contenido del registro T y el resultado es almacenado en el registro P.

***MPYK*** : Multiplicación inmediata.
***Descripción*** : El contenido del registro T es multiplicado con una constante con signo de 13 bits y el resultado es almacenado en el registro P.

***NOP*** : No operación.
***Código*** = 7F80 h.
***Descripción*** : No ejecuta ninguna operación.

***OR*** : Or con los bits de orden bajo del acumulador.
***Código*** = 7AXX h.
***Descripción*** : Se realiza la operación OR con la parte baja del acumulador y el contenido de la memoria de datos; y la parte alta del acumulador OR cero y esto concatenado se guarda en el acumulador.

***OUT*** : Salida de datos por el puerto.
***Código*** = 4XXX h.
***Descripción*** : La instrucción OUT transfiere los datos desde la memorisa de datos a un periferico externo.

***PAC*** : Carga el acumulador con el registro P.
***Código*** = 7F8E h.
***Descripción*** :El contenido del registro P, resultado de la multiplicación es cargado en el acumulador.

***POP*** : POP del tope de la pila al acumulador.
***Código*** = 7F9D h.
***Descripción*** : El contenido del tope de la pila es cargado al acumulador. El siguiente elemento de la pila se convierte en el nuevo tope de la pila.

***PUSH*** : Push del acumulador a la pila.
***Código*** = 79FC h.
***Descripción*** : El contenido de los 12 bits más bajos del acumulador son almacenados en el tope de la pila.

***RET*** : Regreso de subritina.
***Código*** = 7F8D h.
***Descripción*** : El elemento tope de la pila es sacado y cargado en el PC.

***ROVM*** : Resetea el registro de overflow.
***Código*** = 7F8A h.
***Descripción*** : Esta instrucción limpia el modo overflow.

***SACH*** : Almacena la parte alta del acumulador con corrimiento.
***Código*** = 5XXX.
***Descripción*** : Almacena la parte alta del acumulador en la memoria de datos con corrimiento; este corrimiento es especificado en los bits del 8 al 10 , y puede ser de 0, 1, o 4 .

***SACL*** : Almacena parte baja del acumulador.
***Código*** = 50XX h.
***Descripción*** : Almacena los bits de orden bajo del acumulador en la memoria de datos.

***SAR*** : Almacenamiento del registro auxiliar.
***Código*** = 3XXX h.
***Descripción*** : El contenido del registro auxiliar designado es almacenado en la memoria de datos especificada.

***SOVM*** : Setea el registro de modo overflow.
***Código*** = 7F8B h.
***Descripción*** : Pone un 1 en la bandera de OVM del registro de estado. Bit número 14.

*SPAC* : Sustrae el registro P del acumulador.
***Código*** = 7F90 h.
***Descripción*** : El contenido del registro P es restado del contenido del acumulador y el resultado es almacenado en el acumulador.

*SST* : Almacena el registro de datos .
***Código*** = 7C00 h.
***Descripción*** : Los bits de estado son salvados en una dirección de memoria de datos especificada dentro de la página número 1.

*SUB* : Substrae del acumulador un valor con dezp.
***Código*** = 1XXX h .
***Descripción*** : Al contenido de la dirección de memoria se le realiza un dezplazamiento a la izquierda ( este indicado en los bits de codigo del 8 al 11 y puede ser de 0 a 15 ) y substraido del acumulador Durante el corrimiento, los bits de orden bajo del dato son rellenados con cero y los de orden alto con el bit de signo extendido. Y el resultado es almacenado en el acumulador.

*SUBC* : Substracción condicional.
***Código*** : 64XX h.
***Descripción*** : Esta instrucción realiza una resta condicional

*SUBH* : Sustrae de la parte alta del acumulador.
***Código*** : 62XX h.
***Descripción*** : Sustre el contenido de la dirección de memoria de datos especificada de la parte alta del acumulador. El resultado es almacenado en el acumulador.

*SUBS* : Sustre de la parte baja del acumulador con el signo suprimido.
***Código*** : 63XX h.
***Descripción*** : Sustrae el contenido de la dirección de memoria del acumulador con el signo suprimido.

*TBLR* : Lee Tabla.
***Código*** = 67XX h.
***Descripción*** : Esta instrucción transfiere una palabra de dondequiera en memoria de programa ; (es decir ROM interna, ROM externa, RAM externa ) a la localidad especificada de la memoria de datos.

***TBLW*** : Escribe tabla.
***Código*** = 7DXX h.
***Descripción*** : Esta instrucción transfiere una palabra de la localidad de la memoria de datos a una locaalidad externa de la RAM.

***XOR*** : Or exclusiva con los bits de orden bajo del acumulador.
***Código*** = 78XX h.
***Descripción*** : Se realiza la operación OR-exclusiva de la siguiente manera : la parte baja del acumulador y el contenido de la memoria de datos; y la parte alta del acumulador con cero y esto concatenado se guarda en el acumulador.

***ZAC*** : Cero en el acumulador.
***Código*** = 7F89 h.
***Descripción*** : El acumulador es limpiado, poniendo ceros en el.

***ZALH*** : Cero al acumulador y carga parte alta.
***Código*** = 6500 h.
***Descripción*** : Esta instrucción limpia al acumulador y carga el contenido de la dirección de memoria de datos especificado en la parte alta del acumulador.

# TMS32010
# MANUAL DE USUARIO

EL SIMULADOR DEL CO-PROCESADOR TMS32010 ESTA COMPUESTO POR UN PROGRAMA LLAMADO TMS32010.EXE EL CUAL EJECUTA O PONE EN FUNCIONAMIENTO AL SIMULADOR, ES DECIR, SE DEBE TECLEAR :

**C:\>TMS32010 <ENTER>**

UNA VEZ INVOCADO EL SISTEMA PEDIRA QUE SE ESCRIBA EL NOMBRE DEL ARCHIVO DE DONDE TOMARA EL CODIGO QUE PROCESARA, ES DECIR , EL CONJUNTO DE INSTRUCCIONES QUE ESTAN EN UN FORMATO IGUAL AL SIGUIENTE :

```
K0000NO$IDT90000BF900B0004B0090B0006B6E00B7F89B7E02B6700B4A007F216FNO$IDT 1
B7E03B6700B4900BF600B0010BF900B000CB4000B4800BF900B000C7F3E9F       NO$IDT 2
: NO$IDT    02-10-93  17:36:41    ASM32010  PC2.1 84.107             NO$IDT 3
```

DESPUES DE TECLEAR EL NOMBRE DEL ARCHIVO APARECERAN LAS VENTANAS QUE COMPONEN AL SIMULADOR Y EN LA PARTE INFERIOR DE LA PANTALLA APARECE UNA LINEA COMO :

**F3.Inst./(R)egs.Gen./F2-Regs.Edo./(M)em.Prog./(L)oc.Datos/(C)ont. PgUp PgDr**

LA CUAL INDICA QUE TECLAS ESTAN ACTIVADAS PARA EL FUNCIONEMIENTO DEL SISTEMA Y LAS CUALES SE DESCRIBEN :

**F3.Inst.-** ESTA OPCION (F3) SIRVE PARA MODIFICAR ALGUNA PARTE DE LA INSTRUCCION ES DECIR :

(**D**)MA - DIRECCION DE LA MEMORIA
(**I**)NST - LA INSTRUCCION EN EJECUCION
(**M**)OD - MODO DE DIRECCIONAMIENTO, ES DECIR , SI ES (**D**)IRECTO O (**I**)NDIRECTO

**(R)egs.Gen.-** ESTA OPCION (R) SIRVE PARA PODER MODIFICAR ALGUNO DE LOS REGISTROS GENERALES DEL SIMULADOR, ES DECIR, ALGUNO DE LOS REGISTROS COMO SON :

**ACC** - ACUMULADOR
**ARO Y AR1** - REGISTROS DE PROPOSITO GENERAL
**P** - REGISTRO DE PROPOSITO GENERAL
**T** - REGISTRO DE PROPOSITO GENERAL
**PC** - CONTADOR DEL PROGRAMA

**F2-Regs.Edo.-** ESTA OPCION (**F2**) SIRVE PARA MODIFICAR LOS REGISTROS DE ESTADO QUE COMPONEN AL SIMULADOR :

**OV** -REGISTRO QUE INDICA EL SOBREFLUJO DEL ACUMULADOR (1 =NO ó 0 =SI)
**OVM** -REGISTRO DE MODO DE SOBREFLUJO (0=DESHABILITADO ó 1=HABILITADO)
**ARP** -REGISTRO PARA SABER QUE REGISTRO SE ESTA USANDO (AR0 O AR1)
**INTM** -REGISTRO DE INTERRUPCION(0=HABILITADA ó 1=DESHABILITADO)
**DP** -REGISTRO APUNTADOR A LA PAGINA DE MEMORIA DE DATOS (0 ó 1)

**(M)em.Prog.-** ESTA OPCION (**M**) SIRVE PARA PODE MODIFICAR ALGUNA CELDA DE LA MEMORIA DEL PROGRAMA, SOLO CON INDICAR LA DIRECCION DE ESA CELDA DE MEMORIA Y LUEGO PEDIRA EL DATO QUE SERA INTRODUCIDO EN ESA DIRECCION

ES IMPORTANTE RECALCAR QUE UNA VEZ CAPTURADO EL DATO DESEADO LA PANTALLA REGRESARA A DONDE APUNTA EL **PC** PARA CONTINUAR EJECUTANDO EL PROGRAMA.

**(L)oc.Datos-** ESTA OPCION (**L**) SIRVE PARA PODER MODIFICAR ALGUNA CELDA DE LA MEMORIA DE DATOS Y AL IGUAL QUE EN LA MEMORIA DE PROGRAMA SOLO SERA NECESARIO INDICAR LA DIRECCION DE LA MEMORIA QUE SE DESEA MODIFICAR Y LUEGO SE PEDIRA EL DATO QUE QUEDARA EN ESA CELDA DE LA MEMORIA

ES IMPORTANTE RECALCAR QUE UNA VEZ INTRODUCIDO EL DATO DESEADO LA PANTALLA DE LA MEMORIA AFECTADO PERMANECERA AHI HASTA QUE SE DESEE MODIFICAR OTRA CELDA.

**(C)ont.-** ESTA OPCION (**C**) SIRVE PARA IR EJECUTANDO LA INSTRUCCION QUE SE ENCUENTRA EN LA PARTE SUPERIOR DE LA VENTANA DE LA MEMORIA DEL PROGRAMA Y CONFORME SE PRESIONE ESTA TECLA SE EJECUTARA INTRUCCION TRAS INTRUCCION.

CON LAS TECLAS DE **PgUp**, **PgDn** Y CON LAS FLECHAS **Arriba**, **Abajo** NOS PODREMOS MOVER LIBREMENTE DENTRO DE LA PANTALLA DE LA MEMORIA DEL PROGRAMA.

PARA SALIR DEL SISTEMA SOLO ES NECESARIO PRESIONAR LA TECLA DE <**ESC**>.

# Manual Técnico sobre el Movimiento en Ventanas

La Función **Interface** controla los movimientos sobre las ventanas a traves de la lectura de la tecla que se presione en ese momento. Esto es posible de realizar por medio de la lectura de teclas de la Función **lee_tecla** la cual lee la tecla que se encuentra en el Buffer de teclado.

Una vez leida la tecla se procede a desifrar si el código de la tecla existe entre los ya definidos previamente de manera global. Si el código de la tecla esta definida, entonces, a traves de un **Case** realizamos la acción correspondiente al código desifrado.

Antes de que se lea cualquier tecla el sistema despliega las ventanas que componen la presentación de Memoria del Programa, Memoria de datos, Pila, Registros de Estado, Registros Generales y Componentes de la Instrucción a Procesarse en ese momento ver dibujo de pantalla.

Los códigos de las teclas son :

| | | |
|---|---|---|
| #define UP_ARROW | 0x4800 | Flecha hacia arriba |
| #define DN_ARROW | 0x5000 | Flecha hacia abajo |
| #define PG_UP | 0x4900 | Pagina arriba |
| #define PG_DN | 0x5100 | Pagina abajo |
| #define F1 | 0x3b00 | |
| #define F2 | 0x3c00 | Modifica registros de Estado |
| #define F3 | 0x3d00 | Modifica componentes de Instrucción |

| | |
|---|---|
| 0x324d y 0x326d | Teclas m y M para modificar alguna localidad de la Memoria del Programa |
| 0x264c y 0x266c | Teclas l y L para modificar alguna localidad de la memoria de datos |
| 0x1352 y 0x1372 | Teclas r y R para modificar algun Registro General |
| 0x2e43 y 0x2e63 | Teclas de c y C para continuar ejecutando la siguiente instrucción |
| 0x011b | Tecla de ESC para terminar el programa |

Los cuales son clasificados en un **Case** para despues llamar a la función correspondiente y realizar la acción que sea necesaria. En el caso de que se presione alguna tecla que no se este definida sonara un Beep provocado por la función con el mismo nombre.

## Estructura que componen al Sistema

La Pila de Datos esta compuesta de un arreglo de 4 posiciones enteras y 2 enteros (tope, base) los cuales funcionan como contadores dentro de arreglo para marcar la base de la Pila y el Tope de la Pila. Cuando un dato es introducido a la Pila primeramente se verifica que la Pila no este llena ya que de lo contrario se perdera el dato que se localiza en la base de la Pila para luego recorrer los datos una posición hacia la base de la Pila en una posición y asi poder introducir en el tope de la Pila el nuevo dato.

Como podemos observar el movimiento de la Pila no es presisamente el de una Pila normal sino por el contrario actua como una "cola circular", es decir , cuando el frente es igual al posterior el posterior se recorre al igual que el frente para meter el nuevo dato.

Tanto la Memoria de datos como la Memoria del Programa son arreglos de enteros **DATA MEM** y **MEM PROG** los cuales son mostrados en pantalla en formato Hexadecimal.

Las 6 ventanas del Simulador son definidas en un arreglo Bidimensional de 7x4 llamada **Ven** y el cual contiene en cada unoa de sus celdas las coordenadas de cada una de las ventanas presentadas en el simulador y que gracias a la función **pantalla**, **marco** y **sel ventana** Imprimimos en monitor para asi dar la Impresion de ventanas independientes.

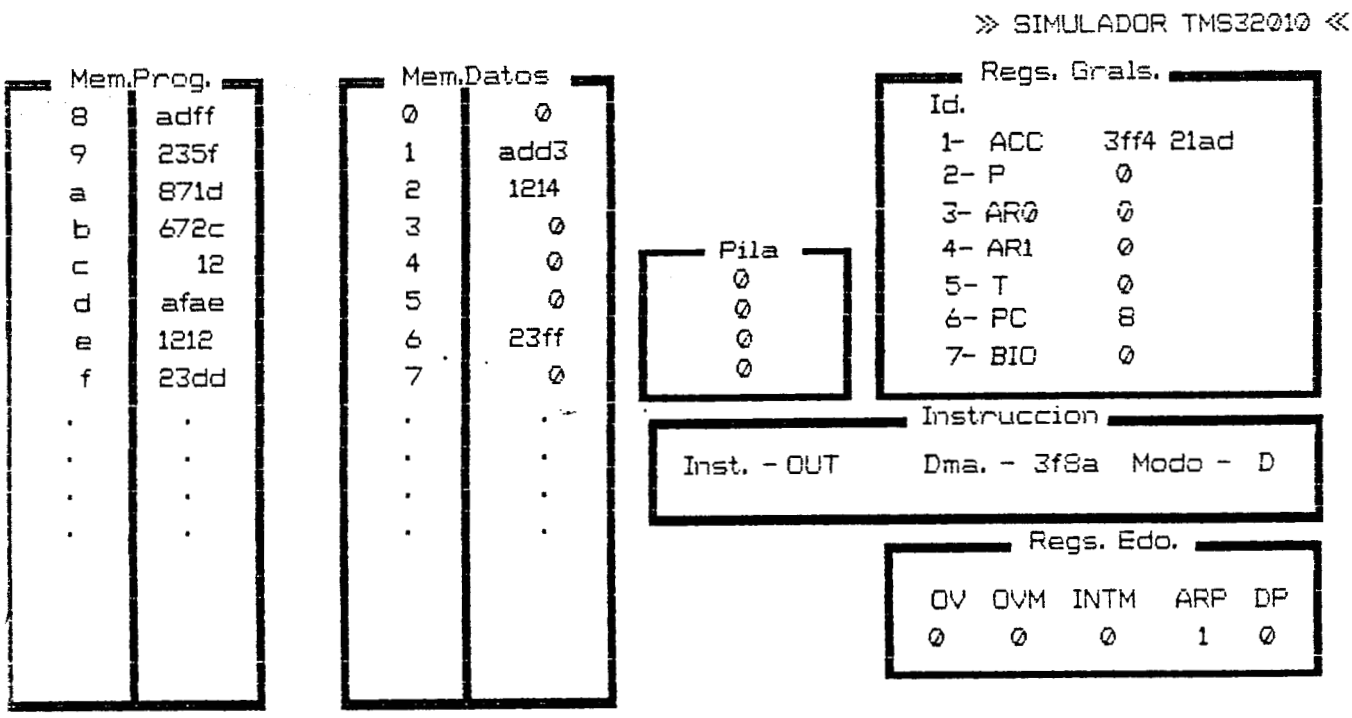
## Funcionemiento General de las Ventanas

Una vez pintados los recuadros de cada ventana tratamos a cada ventana por separado ya sea para colocar los datos internos a ella o para actualizar a alguno de ellos solamente hacemos referencia del arreglo **Ven**. Esto en realidad se logra por que cada ventana en realidad funciona de manera independiente y con solo mencionar que número de ventana se desea afectar y mandandolo como parametro de la función sel_ventana() actuara cualquier efecto de posición e impresión sobre la misma.

Para poder modificar o moverse sobre alguna de las 2 Memoria que existen basta con indicar L ó M y luego indicar que localidad de memoria se desea modificar para que la rutina de despliega() con coloque dentro de las 19 posiciones que se alcanzan a ver en el monitor. Para el caso de la Memoria del Programa una vez modificada la dirección deseada se regresara automaticamente a la posición que le indique el PC ya que la instrucción que se encuentra en la parte superior de la ventana de la Memoria de Programa es la instrucción a ejecutarse y para el caso de la Memoria de Datos una vez que se modifica alguna localidad de la

misma se queda ahi para que el usuario tenga una visión de la celda que modifico.

El programa TMS32010.EXE funciona en cualquier computadora compatible con un minimo de 640 KB en Ram y con monitor VGA,CGA ó TTL. Se pedira de preferencia que el programa recida en el disco duro de la máquina.




F3. Inst./(R)egs. Gen./F2-Regs. Edo./(M)em. Prog./(L)oc. Datos/(C)ont. PgUp PgDn ↑ ↓

Comando :

```
ine  Source
---- ------------------------------------------------------------------------------------------------------------------------
   - //.....................................................................
   - //
   - //   Program Name: TMS32010.C        Copyright: Universidad Autonoma Metropolitana
   - //   Date Created: 07/07/93           Language: Borland C 2.0
   - //   Time Created: 23:10:41             Author: Alberto Garcia Castillo
   - //   Description : Programa principal del Simulador del Co-procesador matematico TMS32010
   - //.....................................................................
   -
   - #include "stdio.h"
0  - #include "dos.h"
1  - #include "ctype.h"
2  - #include "stdlib.h"
3  - #include "c:\tms32010\var.c"       /* Definicion de variables globales       */
4  - #include "c:\tms32010\TMS.C"       /* Procedimientos de manejo de ventanas   */
5  - #include "c:\tms32010\pantalla.c"  /* Pantalla de presentacion del programa  */
6  - FILE *fopen();
7  - FILE *fp;
8  - char nomb_arch[40];
9  -
0  - /* ---------------------- PRINCIPAL ------------------------------ */
1  -
2  - main ()
3  - {
4  -   unsigned int i,instruccion=0;
5  -   comenzar();
6  -   carga_arch();
7  -   pantalla();
8  -   despliega2(pos);
9  -   interface();
0  -   for (i=0; i<=1024; i++)
1  -    {
2  -     instruccion=MEM_PROG[PC];
3  -     interface();
4  -     Tabla(instruccion);
5  -    }
6  - }
7  -
8  - /* ---------------------------------------------------------------- */
9  -
0  - Tabla(instruccion)
1  - unsigned instruccion;
2  -  {
3  -   unsigned int op=0;
4  -
5  -   op=(instruccion & 0xf000);
6  -   switch (op)
7  -    {
8  -     case 0x0    : ADD(instruccion);
9  -     break;
0  -     case 0x1000 : SUB(instruccion);
1  -     break;
2  -     case 0x2000 : LAC(instruccion);
```

```
ine  Source
---- ------------------------------------------------------------------------------------------------------------------------
3  -     break;
4  -     case 0x3000 : op=(instruccion & 0xf00);
5  -     switch (op)
6  -     {
7  -       case 0x0   :
8  -       case 0x100 : SAR(instruccion);
9  -      break;
0  -       case 0x800 :
1  -       case 0x900 : LAR(instruccion);
2  -      break;
3  -     }
4  -     break;
5  -     case 0x4000 : op=(instruccion & 0xf00);
6  -     switch (op)
7  -     {
8  -       case 0x0   :
9  -       case 0x100 :
0  -       case 0x200 :
1  -       case 0x300 :
2  -       case 0x400 :
3  -       case 0x500 :
4  -       case 0x600 :
5  -       case 0x700 : IN(instruccion);
6  -      break;
7  -       case 0x800 :
8  -       case 0x900 :
9  -       case 0xa00 :
0  -       case 0xb00 :
1  -       case 0xc00 :
2  -       case 0xd00 :
3  -       case 0xe00 :
4  -       case 0xf00 : OUT(instruccion);
5  -      break;
6  -     }
7  -     break;
8  -     case 0x5000 : op=(instruccion & 0xf00);
9  -     switch (op)
0  -     {
1  -       case 0x0   : SACL(instruccion);
2  -      break;
3  -       case 0x800 :
4  -       case 0x900 :
5  -       case 0xa00 :
6  -       case 0xb00 :
7  -       case 0xc00 :
8  -       case 0xd00 :
9  -       case 0xe00 :
00 -       case 0xf00 : SACH(instruccion);
01 -      break;
02 -     }
03 -     break;
04 -     case 0x6000 : op=(instruccion & 0xf00);
```

```
Line  Source
----- ----------------------------------------------------------------------------------------------------
105 -     switch (op)
106 -     {
107 -       case 0x0   : ADDH(instruccion);
108 -      break;
109 -       case 0x100 : ADDS(instruccion);
110 -      break;
111 -       case 0x200 : SUBH(instruccion);
112 -      break;
113 -       case 0x300 : SUBS(instruccion);
114 -      break;
115 -       case 0x400 : SUBC(instruccion);
116 -      break;
117 -       case 0x500 : ZALH(instruccion);
118 -      break;
119 -       case 0x600 : ZALS(instruccion);
120 -      break;
121 -       case 0x700 : TBLR(instruccion);
122 -      break;
123 -       case 0x800 :op=(instruccion & 0x0f0);
124 -     switch (op)
125 -     {
126 -       case 0x80:op=(instruccion & 0xf);
127 -          switch (op)
128 -          {
129 -            case 0x0 :
130 -            case 0x1 : LARP(instruccion);
131 -          break;
132 -            case 0x8 :
133 -            case 0x9 : MAR(instruccion);
134 -        break;
135 -          }
136 -          break;
137 -       case 0x90: MAR(instruccion);
138 -           break;
139 -     }
140 -     break;
141 -       case 0x900 : DMOV(instruccion);
142 -      break;
143 -       case 0xa00 : LT(instruccion);
144 -      break;
145 -       case 0xb00 : LTD(instruccion);
146 -      break;
147 -       case 0xc00 : LTA(instruccion);
148 -      break;
149 -       case 0xd00 : MPY(instruccion);
150 -      break;
151 -       case 0xe00 : LDPK(instruccion);
152 -      break;
153 -       case 0xf00 : LDP(instruccion);
154 -      break;
155 -     }
156 -     break;
```

```
le  Source
--- ----------------------------------------------------------------------------
7 -     case 0x7000 : op=(instruccion & 0xf00);
8 -     switch (op)
9 -     {
0 -       case 0x0   :
1 -       case 0x100 : LARK(instruccion);
2 -       break;
3 -       case 0x800 : XOR(instruccion);
4 -      break;
5 -       case 0x900 : AND(instruccion);
6 -      break;
7 -       case 0xa00 : OR(instruccion);
8 -      break;
9 -       case 0xb00 : LST(instruccion);
0 -      break;
1 -       case 0xc00 : SST(instruccion);
2 -      break;
3 -       case 0xd00 : TBLW(instruccion);
4 -      break;
5 -       case 0xe00 : LACK(instruccion);
6 -      break;
7 -       case 0xf00 :op=(instruccion & 0x0f0);
8 -     switch (op)
9 -     {
0 -       case 0x80:op=(instruccion & 0xf);
1 -          switch (op)
2 -          {
3 -            case 0x0 : CLAVE=36; /* NOP */
4 -         break;
5 -            case 0x1 : DINT();
6 -         break;
7 -            case 0x2 : EINT();
8 -         break;
9 -            case 0x8 : ABS(instruccion);
0 -         break;
1 -            case 0x9 : ZAC();
2 -         break;
3 -            case 0xa : ROVM();
4 -         break;
5 -            case 0xb : SOVM();
6 -         break;
7 -            case 0xc : CALA();
8 -         break;
9 -            case 0xd : RET();
0 -         break;
1 -            case 0xe : PAC();
2 -         break;
3 -            case 0xf : APAC();
4 -         break;
5 -          }
6 -          break;
7 -       case 0x90:op=(instruccion & 0xf);
8 -          switch (op)
```

```
Line  Source
----- ----------------------------------------------------------------------------------------------------
209 -          {
210 -            case 0x0   : SPAC();
211 -           break;
212 -            case 0xc   : PUSH();
213 -           break;
214 -            case 0xd   : POP();
215 -           break;
216 -           }
217 -           break;
218 -     }
219 -     break;
220 -     }
221 -     break;
222 -     case 0x8000 :
223 -     case 0x9000 : MPYK(instruccion);
224 -     break;
225 -     case 0xf000 :op=(instruccion & 0xf00);
226 -     switch (op)
227 -     {
228 -       case 0x400 :PC++;
229 -     instruccion=MEM_PROG[PC];
230 -     BANZ(instruccion);
231 -      break;
232 -       case 0x500 :PC++;
233 -     instruccion=MEM_PROG[PC];
234 -     BV(instruccion);
235 -      break;
236 -       case 0x600 :PC++;
237 -     instruccion=MEM_PROG[PC];
238 -     BIOZ(instruccion);
239 -      break;
240 -       case 0x800 : break;
241 -       case 0x900 : PC++;
242 -     instruccion=MEM_PROG[PC];
243 -     B(instruccion);
244 -      break;
245 -       case 0xa00 :PC++;
246 -     instruccion=MEM_PROG[PC];
247 -     BLZ(instruccion);
248 -      break;
249 -       case 0xb00 :PC++;
250 -     instruccion=MEM_PROG[PC];
251 -     BLEZ(instruccion);
252 -      break;
253 -       case 0xc00 :
254 -     PC++;
255 -     instruccion=MEM_PROG[PC];
256 -     BGZ(instruccion);
257 -      break;
258 -       case 0xd00 :
259 -     PC++;
260 -     instruccion=MEM_PROG[PC];
```

```
Line  Source
----- ----------------------------------------------------------------------------------------------------
 161 -     BGEZ(instruccion);
 162 -      break;
 163 -       case 0xe00 :
 164 -     PC++;
 165 -     instruccion=MEM_PROG[PC];
 166 -     BNZ(instruccion);
 167 -      break;
 168 -       case 0xf00 :
 169 -     PC++;
 170 -     instruccion=MEM_PROG[PC];
 171 -     BZ(instruccion);
 172 -      break;
 173 -     }
 174 -     break;
 175 -     }
 176 -   }
 177 -
 178 - /* ------------------------------------------------------------ */
 179 -
 180 - carga_arch()
 181 - {
 182 -   char c;
 183 -   int abrir, num, instruccion, i;
 184 -   clrscr();
 185 -   abrir = abre_arch(&nomb_arch);
 186 -
 187 -   if (abrir == 0)                    /* si existe */
 188 -   {
 189 -       gotoxy(10,100);
 190 -       printf(" \nLEYENDO EL ARCHIVO %s\n",nomb_arch);
 191 -       fp = fopen(nomb_arch,"r");
 192 -       while ((c=getc(fp))!=EOF)
 193 -        {
 194 -  if (c=='B')
 195 -    {
 196 -     instruccion=0;
 197 -     num=0;
 198 -     for (i=0; i<=3; i++)
 199 -      {
 200 -       c=getc(fp);
 201 -       switch(c)
 202 -         {
 203 -   case '0': num=0;
 204 -      instruccion=((instruccion<<4)|num);
 205 -      break;
 206 -   case '1': num=1;
 207 -      instruccion=((instruccion<<4)|num);
 208 -      break;
 209 -   case '2': num=2;
 210 -      instruccion=((instruccion<<4)|num);
 211 -      break;
 212 -   case '3': num=3;
```

```
Line  Source
----- ----------------------------------------------------------------------------------------------------
513 -      instruccion=((instruccion<<4)|num);
514 -      break;
515 -   case '4': num=4;
516 -      instruccion=((instruccion<<4)|num);
517 -      break;
518 -   case '5': num=5;
519 -      instruccion=((instruccion<<4)|num);
520 -      break;
521 -   case '6': num=6;
522 -      instruccion=((instruccion<<4)|num);
523 -      break;
524 -   case '7': num=7;
525 -      instruccion=((instruccion<<4)|num);
526 -      break;
527 -   case '8': num=8;
528 -      instruccion=((instruccion<<4)|num);
529 -      break;
530 -   case '9': num=9;
531 -      instruccion=((instruccion<<4)|num);
532 -      break;
533 -   case 'A': num=10;
534 -      instruccion=((instruccion<<4)|num);
535 -      break;
536 -   case 'B': num=11;
537 -      instruccion=((instruccion<<4)|num);
538 -      break;
539 -   case 'C': num=12;
540 -      instruccion=((instruccion<<4)|num);
541 -      break;
542 -   case 'D': num=13;
543 -      instruccion=((instruccion<<4)|num);
544 -      break;
545 -   case 'E': num=14;
546 -      instruccion=((instruccion<<4)|num);
547 -      break;
548 -   case 'F': num=15;
549 -      instruccion=((instruccion<<4)|num);
550 -      break;
551 -         }
552 -      }
553 -     MEM_PROG[CONT]=instruccion;
554 -     ++CONT;
555 -    }
556 -        }
557 -   }
558 - }
559 -
560 - /* ------------------------------------------------------------- */
561 -
562 - abre_arch(char nomb_arch[40])
563 - {
564 -   FILE *infile;
```

```
ine  Source
---- ------------------------------------------------------------------------------------------------------------------------
55 -   int encuentra;
56 -   do {
57 -       clrscr();
58 -       gotoxy(10,50);
59 -       printf( "DAME EL ARCHIVO A ANALIZAR : " );
70 -       gets(nomb_arch);
71 -       if ( !strlen(nomb_arch) )
72 -    encuentra = 1;
73 -       else
74 -         {
75 -     infile = fopen( nomb_arch, "r" );
76 -            if ( !infile )
77 -               {
78 -    printf( "NO EXISTE EL ARCHIVO !\n" );
79 -    delay(2000);
80 -    encuentra = 2;
81 -                }
82 -            else
83 -   encuentra = 0;
84 -            fclose(infile);        /* CIERRA EL ARCHIVO */
85 -         }
86 -     }while ((encuentra != 0)&&(encuentra != 1));
87 -    return(encuentra);
88 - }
89 -
90 - /* ------------------------------------------------------------ */
91 -
92 - DIR_IND(instruccion)
93 - unsigned int instruccion;
94 - {
95 -   if (instruccion & 0x20)
96 -     REG_AUX[ARP]++;
97 -   if (instruccion & 0x10)
98 -     REG_AUX[ARP]--;
99 -   if ((instruccion & 0x08)==0)
100 -    {
101 -     ARP=(instruccion & 0x01);
102 -     ESTADO=status();
103 -    }
104 -   bandera=3;
105 - }
106 -
107 - /* ------------------------------------------------------------ */
108 -
109 - ADD(instruccion)
110 - unsigned int instruccion;
111 - {
112 -  unsigned int shift;
113 -    shift = (instruccion & 0xf00);
114 -    shift = shift>>8;
115 -    mod_dir=(instruccion & 0x80);
116 -    if (mod_dir==0)
```

```
ine  Source
---- ------------------------------------------------------------------------------------------------------------
17 -     {
18 -      dma=(instruccion & 0x7f);
19 -      ACC=ACC+(DATA_MEM[dma]<<shift);
20 -      bandera=2;
21 -     }
22 -    else
23 -     {
24 -       ACC=ACC+(DATA_MEM[REG_AUX[ARP]]<<shift);
25 -       DIR_IND(instruccion);
26 -     }
27 -     CLAVE=1;
28 -     PC++;
29 - }
30 -
31 - /* ------------------------------------------------------------ */
32 -
33 - ADDH(instruccion)
34 - unsigned int instruccion;
35 - {
36 -  long int dato;
37 -    mod_dir=(instruccion & 0x80);
38 -    if (mod_dir==0)
39 -     {
40 -      dma=(instruccion & 0x7f);
41 -      dato= (DATA_MEM[dma]<<16);
42 -      ACC=ACC+dato;
43 -      bandera=2;
44 -     }
45 -    else
46 -     {
47 -       dato=(DATA_MEM[REG_AUX[ARP]]<<16);
48 -       ACC=ACC+dato;
49 -       DIR_IND(instruccion);
50 -     }
51 -     CLAVE=2;
52 -     PC++;
53 - }
54 -
55 - /* ------------------------------------------------------------ */
56 -
57 - ADDS(instruccion)
58 - unsigned int instruccion;
59 - {
60 -  long int dato;
61 -    mod_dir=(instruccion & 0x80);
62 -    if (mod_dir==0)
63 -     {
64 -      dma=(instruccion & 0x7f);
65 -      dato= (DATA_MEM[dma] &  0xffff);
66 -      ACC=ACC+dato;
67 -      bandera=2;
68 -     }
```

```
ine  Source
---- ------------------------------------------------------------------------------------------------------------------------
59 -    else
70 -     {
71 -       dato=(DATA_MEM[REG_AUX[ARP]] & 0xffff);
72 -       ACC=ACC+dato;
73 -       DIR_IND(instruccion);
74 -     }
75 -     CLAVE=3;
76 -     PC++;
77 - }
78 -
79 - /* ------------------------------------------------------------ */
80 -
81 - AND(instruccion)
82 - unsigned int instruccion;
83 - {
84 -    mod_dir=(instruccion & 0x80);
85 -    if (mod_dir==0)
86 -     {
87 -      dma=(instruccion & 0x7f);
88 -      ACC=(ACC & 0x0000ffff);
89 -      ACC=(ACC & DATA_MEM[dma]);
90 -      bandera=2;
91 -     }
92 -    else
93 -      {
94 -       ACC=(ACC & 0x0000ffff);
95 -       ACC=(ACC & REG_AUX[ARP]);
96 -       DIR_IND(instruccion);
97 -      }
98 -     CLAVE=4;
99 -     PC++;
00 - }
01 -
02 - /* ------------------------------------------------------------ */
03 -
04 - APAC()
05 - {
06 -   ACC=ACC+P;
07 -   CLAVE=5;
08 -   PC++;
09 -   bandera=5;
10 - }
11 -
12 - /* ------------------------------------------------------------ */
13 -
14 - B(instruccion)
15 - unsigned int instruccion;
16 - {
17 -   unsigned int pma;
18 -   pma=(instruccion & 0xfff);
19 -   PC=pma;
20 -   CLAVE=6;
```

```
ine  Source
---- ------------------------------------------------------------------------------------------------------------------------
21 -   bandera=5;
22 - }
23 -
24 - /* ------------------------------------------------------------- */
25 -
26 - BANZ(instruccion)
27 - unsigned int instruccion;
28 - {
29 -   unsigned int pma;
30 -   pma=(instruccion & 0xfff);
31 -   if((REG_AUX[ARP] & 0x1ff)!=0)
32 -    {
33 -     REG_AUX[ARP]=REG_AUX[ARP]-1;
34 -     PC=pma;
35 -    }
36 -   else
37 -    {
38 -     REG_AUX[ARP]=REG_AUX[ARP]-1;
39 -     PC++;
40 -    }
41 -   CLAVE=7;
42 -   bandera=5;
43 - }
44 -
45 - /* ------------------------------------------------------------ */
46 -
47 - BGEZ(instruccion)
48 - unsigned int instruccion;
49 - {
50 -   unsigned int pma;
51 -   pma=(instruccion & 0xfff);
52 -   if(ACC>=0)
53 -     PC=pma;
54 -   else
55 -     PC++;
56 -   CLAVE=8;
57 -   bandera=5;
58 - }
59 -
60 - /* ------------------------------------------------------------ */
61 -
62 - BGZ(instruccion)
63 - unsigned int instruccion;
64 - {
65 -   unsigned int pma;
66 -   pma=(instruccion & 0xfff);
67 -   if(ACC>0)
68 -     PC=pma;
69 -   else
70 -     PC++;
71 -   CLAVE=9;
72 -   bandera=5;
```

```
Line  Source
---- ------------------------------------------------------------------------------------------------------------------------
473 - }
474 -
475 - /* ------------------------------------------------------------ */
476 -
477 - BIOZ(instruccion)
478 - unsigned int instruccion;
479 - {
480 -   unsigned int pma;
481 -   pma=(instruccion & 0xfff);
482 -   if(BIO==0)
483 -     PC=pma;
484 -   else
485 -     PC++;
486 -   CLAVE=10;
487 -   bandera=5;
488 - }
489 -
490 - /* ------------------------------------------------------------ */
491 -
492 - BLEZ(instruccion)
493 - unsigned int instruccion;
494 - {
495 -   unsigned int pma;
496 -   pma=(instruccion & 0xfff);
497 -   if(ACC<=0)
498 -     PC=pma;
499 -   else
500 -     PC++;
501 -   CLAVE=11;
502 -   bandera=5;
503 - }
504 -
505 - /* ------------------------------------------------------------ */
506 -
507 - BLZ(instruccion)
508 - unsigned int instruccion;
509 - {
510 -   unsigned int pma;
511 -   pma=(instruccion & 0xfff);
612 -   if(ACC<0)
513 -     PC=pma;
514 -   else
515 -     PC++;
616 -   CLAVE=12;
517 -   bandera=5;
618 - }
519 -
620 - /* ------------------------------------------------------------ */
521 -
622 - BNZ(instruccion)
623 - unsigned int instruccion;
624 - {
```

```
Line  Source
----- ----------------------------------------------------------------------------------------------------
525 -   unsigned int pma;
526 -   pma=(instruccion & 0xfff);
527 -   if(ACC!=0)
528 -     PC=pma;
529 -   else
530 -     PC++;
531 -   CLAVE=13;
532 -   bandera=5;
533 - }
534 -
535 - /* ------------------------------------------------------------ */
536 -
537 - BV(instruccion)
538 - unsigned int instruccion;
539 - {
540 -   unsigned int pma;
541 -   pma=(instruccion & 0xfff);
542 -   if(OV==1)
543 -    {
544 -     PC=pma;
545 -     OV=0;
546 -    }
547 -   else
548 -     PC++;
549 -   ESTADO=status();
550 -   CLAVE=14;
551 -   bandera=5;
552 - }
553 -
554 - /* ------------------------------------------------------------ */
555 -
556 - BZ(instruccion)
557 - unsigned int instruccion;
558 - {
559 -   unsigned int pma;
560 -   pma=(instruccion & 0xfff);
561 -   if(ACC==0)
562 -     PC=pma;
563 -   else
564 -     PC++;
565 -   CLAVE=15;
566 -   bandera=5;
567 - }
568 -
569 - /* ------------------------------------------------------------ */
570 -
571 - DINT()
572 - {
573 -   INTM=1;
574 -   ESTADO=status();
575 -   CLAVE=18;
576 -   PC++;
```

```
ine  Source
---- ------------------------------------------------------------------------------------------------------------------------
77 -   bandera=5;
78 - }
79 -
80 - /* ------------------------------------------------------------ */
81 -
82 - DMOV(instruccion)
83 - unsigned int instruccion;
84 - {
85 -    mod_dir=(instruccion & 0x80);
86 -    if (mod_dir==0)
87 -     {
88 -      dma=(instruccion & 0x7f);
89 -      DATA_MEM[dma+1]=DATA_MEM[dma];
90 -      bandera=2;
91 -     }
92 -    else
93 -     {
94 -       DATA_MEM[REG_AUX[ARP]+1]=DATA_MEM[REG_AUX[ARP]] ;
95 -       DIR_IND(instruccion);
96 -     }
97 -     PC++;
98 -   CLAVE=19;
99 - }
00 -
01 - /* ------------------------------------------------------------ */
02 -
03 - EINT()
04 - {
05 -   INTM=0;
06 -   ESTADO=status();
07 -   PC++;
08 -   CLAVE=20;
09 -   bandera=5;
10 - }
11 -
12 - /* ------------------------------------------------------------ */
13 -
14 - LAC(instruccion)
15 - unsigned int instruccion;
16 - {
17 -  unsigned int shift;
18 -    shift = (instruccion & 0xf00);
19 -    shift = shift>>8;
20 -    mod_dir=(instruccion & 0x80);
21 -    if (mod_dir==0)
22 -     {
23 -      dma=(instruccion & 0x7f);
24 -      ACC=(DATA_MEM[dma]<<shift);
25 -      bandera=2;
26 -     }
27 -    else
28 -     {
```

```
Line  Source
---- ------------------------------------------------------------------------------------------------------------------------
'29 -      ACC=(DATA_MEM[REG_AUX[ARP]]<<shift);
'30 -      DIR_IND(instruccion);
'31 -     }
'32 -   CLAVE=22;
'33 -    PC++;
'34 - }
'35 -
'36 - /* ------------------------------------------------------------ */
'37 -
'38 - LACK(instruccion)
'39 - unsigned int instruccion;
'40 - {
'41 -   unsigned int cte=0;
'42 -   cte=(instruccion & 0xff);
'43 -   ACC=(ACC & 0x0);
'44 -   ACC=cte;
'45 -   PC++;
'46 -   CLAVE=23;
'47 -   bandera=4;
'48 - }
'49 -
'50 - /* ------------------------------------------------------------ */
'51 -
'52 - LAR(instruccion)
'53 - unsigned int instruccion;
'54 - {
'55 -  unsigned int  arp_aux;
'56 -    arp_aux= (instruccion & 700);
'57 -    arp_aux=( arp_aux>>8);
'58 -    mod_dir=(instruccion & 0x80);
'59 -    if (mod_dir==0)
'60 -     {
'61 -      dma=(instruccion & 0x7f);
'62 -      REG_AUX[arp_aux]=DATA_MEM[dma];
'63 -      bandera=2;
'64 -     }
'65 -    else
'66 -     {
'67 -      REG_AUX[arp_aux]=DATA_MEM[REG_AUX[arp_aux]];
'68 -      DIR_IND(instruccion);
'69 -     }
'70 -    PC++;
'71 -   CLAVE=24;
'72 - }
'73 -
'74 - /* ------------------------------------------------------------ */
'75 -
'76 - LARK(instruccion)
'77 - unsigned int instruccion;
'78 - {
'79 -  unsigned int cte_aux;
'80 -  arp_aux=(instruccion & 0x700);
```

```
ine  Source
---- ------------------------------------------------------------------------------------------------------------------------
 81 -  arp_aux=(arp_aux>>8);
 82 -  cte_aux=(instruccion & 0xff);
 83 -  REG_AUX[arp_aux]=cte_aux;
 84 -  PC++;
 85 -  CLAVE=25;
 86 -  bandera=6;
 87 - }
 88 -
 89 - /* ------------------------------------------------------------ */
 90 -
 91 - LARP(instruccion)
 92 - unsigned int instruccion;
 93 - {
 94 -  unsigned int cte1;
 95 -  cte1= (instruccion & 0x1);
 96 -  ARP=cte1;
 97 -  ESTADO=status();
 98 -  PC++;
 99 -  CLAVE=26;
100 -  bandera=7;
101 - }
102 -
103 - /* ------------------------------------------------------------ */
104 -
105 - LDP(instruccion)
106 - unsigned int instruccion;
107 - {
108 -  unsigned int   LSB;
109 -    mod_dir=(instruccion & 0x80);
110 -    if (mod_dir==0)
111 -     {
112 -      daa=(instruccion & 0x7f);
113 -      LSB=(0x1 & DATA_MEM[daa]);
114 -      DP=LSB;
115 -      bandera=2;
116 -     }
117 -    else
118 -     {
119 -      LSB=(0x1 & DATA_MEM[REG_AUX[ARP]]);
120 -      DP=LSB;
121 -      DIR_IND(instruccion);
122 -     }
123 -     ESTADO=status();
124 -    PC++;
125 -    CLAVE=27;
126 - }
127 -
128 - /* ------------------------------------------------------------ */
129 -
130 - LDPK(instruccion)
131 - unsigned int instruccion;
132 - {
```

```
Line  Source
----- ----------------------------------------------------------------------------------------------------
133 -   unsigned int ctel=0;
134 -   ctel=(instruccion & 0x1);
135 -   DP=ctel;
136 -   ESTADO=status();
137 -   PC++;
138 -   CLAVE=28;
139 -   bandera=7;
140 - }
141 -
142 - /* ------------------------------------------------------------ */
143 -
144 - LST(instruccion)
145 - unsigned int instruccion;
146 - {
147 -    mod_dir=(instruccion & 0x80);
148 -    if (mod_dir==0)
149 -     {
150 -      dma=(instruccion & 0x7f);
151 -      ESTADO=DATA_MEM[dma];
152 -      bandera=2;
153 -     }
154 -    else
155 -     {
156 -       ESTADO=DATA_MEM[REG_AUX[ARP]] ;
157 -       DIR_IND(instruccion);
158 -     }
159 -     PC++;
160 -     CLAVE=29;
161 - }
162 -
163 - /* ------------------------------------------------------------ */
164 -
165 - LT(instruccion)
166 - unsigned int instruccion;
167 - {
168 -    mod_dir=(instruccion & 0x80);
169 -    if (mod_dir==0)
170 -     {
171 -      dma=(instruccion & 0x7f);
172 -      T=DATA_MEM[dma];
173 -      bandera=2;
174 -     }
175 -    else
176 -     {
177 -      T=DATA_MEM[REG_AUX[ARP]];
178 -      DIR_IND(instruccion);
179 -     }
180 -    PC++;
181 -   CLAVE=30;
182 - }
183 -
184 - /* ------------------------------------------------------------ */
```

```
ine  Source
---- ------------------------------------------------------------------------------------------------------------------------
 85 -
 86 - LTA(instruccion)
 87 - unsigned int instruccion;
 88 - {
 89 -    mod_dir=(instruccion & 0x80);
 90 -    if (mod_dir==0)
 91 -     {
 92 -      dma=(instruccion & 0x7f);
 93 -      T=DATA_MEM[dma];
 94 -      ACC= ACC+P;
 95 -      bandera=2;
 96 -     }
 97 -    else
 98 -     {
 99 -      T=DATA_MEM[REG_AUX[ARP]];
100 -      ACC= ACC+P;
101 -      DIR_IND(instruccion);
102 -     }
103 -    PC++;
104 -   CLAVE=31;
105 - }
106 -
107 - /* ------------------------------------------------------------- */
108 -
109 - LTD(instruccion)
110 - unsigned int instruccion;
111 - {
112 -    mod_dir=(instruccion & 0x80);
113 -    if (mod_dir==0)
114 -     {
115 -      dma=(instruccion & 0x7f);
116 -      T=DATA_MEM[dma];
117 -      DATA_MEM[dma+1]=DATA_MEM[dma];
118 -      ACC= ACC+P;
119 -      bandera=2;
120 -     }
121 -    else
122 -     {
123 -      T=DATA_MEM[REG_AUX[ARP]];
124 -      DATA_MEM[REG_AUX[ARP]+1]=DATA_MEM[REG_AUX[ARP]];
125 -      ACC= ACC+P;
126 -      DIR_IND(instruccion);
127 -     }
128 -    PC++;
129 -   CLAVE=32;
130 - }
131 -
132 - /* ------------------------------------------------------------- */
133 -
134 - MAR(instruccion)
135 - unsigned int instruccion;
136 - {
```

```
ine  Source
---- ------------------------------------------------------------------------------------------------------------------------
37 -  mod_dir=(instruccion & 0x80);
38 -  if(mod_dir==0x80)
39 -     DIR_IND();
40 -  else
41 -    bandera=2;
42 -  PC++;
43 -  CLAVE=33;
44 - }
45 -
46 - /* ------------------------------------------------------------------ */
47 -
48 - MPY(instruccion)
49 - unsigned int instruccion;
50 - {
51 -    mod_dir=(instruccion & 0x80);
52 -    if (mod_dir==0)
53 -     {
54 -      dma=(instruccion & 0x7f);
55 -      P=T*DATA_MEM[dma];
56 -      bandera=2;
57 -     }
58 -    else
59 -     {
60 -      P=T*DATA_MEM[REG_AUX[ARP]];
61 -      DIR_IND(instruccion);
62 -     }
63 -    PC++;
64 -   CLAVE=34;
65 - }
66 -
67 - /* ------------------------------------------------------------------ */
68 -
69 - MPYK(instruccion)
70 - unsigned int instruccion;
71 - {
72 -   int ctes;
73 -   ctes=(instruccion & 0x1fff);
74 -   P=ctes*T;
75 -   PC++;
76 -   CLAVE=35;
77 -   bandera=6;
78 - }
79 -
80 - /* ------------------------------------------------------------------ */
81 -
82 - OR(instruccion)
83 - unsigned int instruccion;
84 - {
85 -    mod_dir=(instruccion & 0x80);
86 -    if (mod_dir==0)
87 -     {
88 -      dma=(instruccion & 0x7f);
```

```
ine  Source
---- ------------------------------------------------------------------------------------------------------------------------
39 -      ACC=(ACC & 0xffff);
70 -      ACC=(ACC | DATA_MEM[dma]);
71 -
72 -     }
73 -    else
74 -     {
75 -       ACC=(ACC & 0xffff);
76 -       ACC=(ACC | DATA_MEM[REG_AUX[ARP]]);
77 -       DIR_IND(instruccion);
78 -     }
79 -    PC++;
000-   CLAVE=37;
001- }
002-
003- /* ------------------------------------------------------------------ */
004-
005- PAC()
006- {
007-   ACC=P;
008-   PC++;
009-   CLAVE=39;
010-   bandera=5;
011- }
012-
013- /* ------------------------------------------------------------------ */
014-
015- ROVM()
016- {
017-  OVM=0;
018-  ESTADO=status();
019-  PC++;
020-  CLAVE=43;
021-  bandera=5;
022- }
023-
024- /* ------------------------------------------------------------------ */
025-
026- SACL(instruccion)
027- unsigned int instruccion;
028- {
029-    mod_dir=(instruccion & 0x80);
030-    if (mod_dir==0)
031-     {
032-      dma=(instruccion & 0x7f);
033-      DATA_MEM[dma]=ACC;
034-      bandera=2;
035-     }
036-    else
037-     {
038-       DATA_MEM[REG_AUX[ARP]]=ACC;
039-       DIR_IND(instruccion);
040-     }
```

```
Line  Source
----- ----------------------------------------------------------------------------------------------------
1041-    PC++;
1042-    CLAVE=45;
1043- }
1044-
1045- /* ------------------------------------------------------------ */
1046-
1047- SOVM()
1048- {
1049-  OVM=1;
1050-  ESTADO=status();
1051-  PC++;
1052-  CLAVE=47;
1053-  bandera=5;
1054- }
1055-
1056- /* ------------------------------------------------------------ */
1057-
1058- SPAC()
1059- {
1060-  ACC=ACC-P;
1061-  PC++;
1062-  CLAVE=48;
1063-  bandera=5;
1064- }
1065-
1066- /* ------------------------------------------------------------ */
1067-
1068- SST(instruccion)
1069- unsigned int instruccion;
1070- {
1071-    mod_dir=(instruccion & 0x80);
1072-    if (mod_dir==0)
1073-     {
1074-      dma=(instruccion & 0x7f);
1075-      DATA_MEM[dma]=status();
1076-      bandera=2;
1077-     }
1078-    else
1079-     {
1080-      DATA_MEM[REG_AUX[ARP]]=status();
1081-      DIR_IND(instruccion);
1082-     }
1083-    PC++;
1084-    CLAVE=49;
1085- }
1086-
1087- /* ------------------------------------------------------------ */
1088-
1089- SUB(instruccion)
1090- unsigned int instruccion;
1091- {
1092-  unsigned int  shift;
```

```
Line  Source
----- ----------------------------------------------------------------------------------------------------------------
1093-    shift = (instruccion & 0xf00);
1094-    shift = shift>>8;
1095-    mod_dir=(instruccion & 0x80);
1096-    if (mod_dir==0)
1097-     {
1098-      dma=(instruccion & 0x7f);
1099-      ACC=ACC-(DATA_MEM[dma]<<shift);
1100-      bandera=2;
1101-     }
1102-    else
1103-     {
1104-       ACC=ACC-DATA_MEM[REG_AUX[ARP]];
1105-       DIR_IND(instruccion);
1106-     }
1107-    PC++;
1108-    CLAVE=50;
1109- }
1110-
1111- /* ------------------------------------------------------------------ */
1112-
1113- SUBS(instruccion)
1114- unsigned int instruccion;
1115- {
1116-    mod_dir=(instruccion & 0x80);
1117-    if (mod_dir==0)
1118-     {
1119-      dma=(instruccion & 0x7f);
1120-      ACC=ACC-DATA_MEM[dma];
1121-      bandera=2;
1122-     }
1123-    else
1124-     {
1125-       ACC=ACC-DATA_MEM[REG_AUX[ARP]];
1126-       DIR_IND(instruccion);
1127-     }
1128-    PC++;
1129-    CLAVE=53;
1130- }
1131-
1132-
1133- /* ------------------------------------------------------------------ */
1134-
1135- TBLR(instruccion)
1136- unsigned int (instruccion);
1137- {
1138-   meter(0xfff & (PC+1));
1139-
1140-   PC= (ACC & 0xfff);
1141-   mod_dir=(instruccion & 0x80);
1142-    if (mod_dir==0)
1143-     {
1144-      dma=(instruccion & 0x7f);
```

```
Line  Source
----- ----------------------------------------------------------------------------------------------------
1145-      DATA_MEM[dma]=MEM_PROG[PC];
1146-      PC=sacar();
1147-      bandera=2;
1148-     }
1149-    else
1150-     {
1151-      DATA_MEM[REG_AUX[ARP]]=MEM_PROG[PC];
1152-      PC=sacar();
1153-      DIR_IND(instruccion);
1154-     }
1155-    CLAVE=54;
1156- }
1157-
1158-
1159- /* ------------------------------------------------------------- */
1160-
1161- TBLW(instruccion)
1162- unsigned int (instruccion);
1163- {
1164-   meter(0xfff & (PC+1));
1165-
1166-   PC= (ACC & 0xfff);
1167-   mod_dir=(instruccion & 0x80);
1168-    if (mod_dir==0)
1169-     {
1170-      dma=(instruccion & 0x7f);
1171-      MEM_PROG[PC]=DATA_MEM[dma];
1172-      PC=sacar();
1173-      bandera=2;
1174-     }
1175-    else
1176-     {
1177-      MEM_PROG[PC]=DATA_MEM[REG_AUX[ARP]];
1178-      PC=sacar();
1179-      DIR_IND(instruccion);
1180-     }
1181-    CLAVE=55;
1182- }
1183-
1184- /* ------------------------------------------------------------- */
1185-
1186- XOR(instruccion)
1187- unsigned int instruccion;
1188- {
1189-    mod_dir=(instruccion & 0x80);
1190-    if (mod_dir==0)
1191-     {
1192-      dma=(instruccion & 0x7f);
1193-      ACC=(ACC & 0xffff);
1194-      ACC=(ACC ^ DATA_MEM[dma]);
1195-      bandera=2;
1196-     }
```

```
Line  Source
----- ----------------------------------------------------------------------------------------------------
 197-    else
 198-     {
 199-       ACC=(ACC & 0xffff);
 200-       ACC=(ACC ^ DATA_MEM[REG_AUX[ARP]]);
 201-       DIR_IND(instruccion);
 202-     }
 203-    PC++;
 204-    CLAVE=56;
 205- }
 206-
 207- /* ------------------------------------------------------------ */
 208-
 209- ZAC()
 210- {
 211-  ACC=0;
 212-  PC++;
 213-  CLAVE=57;
 214-  bandera=5;
 215- }
 216-
 217- /* ------------------------------------------------------------ */
 218-
 219- ZALH(instruccion)
 220- unsigned int instruccion;
 221- {
 222-    mod_dir=(instruccion & 0x80);
 223-    if (mod_dir==0)
 224-     {
 225-      dma=(instruccion & 0x7f);
 226-      ACC=(0xffff & DATA_MEM[dma]);
 227-      ACC=(ACC<<16);
 228-      bandera=2;
 229-     }
 230-    else
 231-     {
 232-      ACC=(0xffff & DATA_MEM[REG_AUX[ARP]]);
 233-      ACC=(ACC<<16);
 234-      DIR_IND(instruccion);
 235-     }
 236-    PC++;
 237-  CLAVE=58;
 238- }
 239-
 240- /* ------------------------------------------------------------ */
 241-
 242- ZALS(instruccion)
 243- unsigned int instruccion;
 244- {
 245-    mod_dir=(instruccion & 0x80);
 246-    if (mod_dir==0)
 247-     {
 248-      dma=(instruccion & 0x7f);
```

```
Line  Source
----- ----------------------------------------------------------------------------------------------------------------
1249-      ACC=(0xffff & DATA_MEM[dma]);
1250-      bandera=2;
1251-     }
1252-    else
1253-     {
1254-      ACC=(0xffff & DATA_MEM[REG_AUX[ARP]]);
1255-      DIR_IND(instruccion);
1256-     }
1257-    PC++;
1258-  CLAVE=59;
1259- }
1260-
1261- /* ------------------------------------------------------------ */
1262-
1263- ABS()
1264- {
1265-   if ((ACC == 0x80000000) && (OVM==1))
1266-      ACC= 0x7fffffff;
1267-   else
1268-     if (ACC<0)
1269-  ACC=-1*ACC;
1270-   PC++;
1271-   CLAVE=0;
1272-   bandera=5;
1273- }
1274-
1275- /* ------------------------------------------------------------ */
1276-
1277- IN(instruccion)
1278- unsigned int instruccion;
1279- {
1280-   unsigned int port_dir;
1281-   port_dir=instruccion & 0x700;
1282-   port_dir=port_dir>>8;
1283-
1284-   gotoxy(1,25);
1285-   cputs("Interrupcion provocada por el Puerto :");
1286-   cprintf("%d ",port_dir);
1287-   cputs(" Valor de entrada :");
1288-   cscanf("%4x",&valor);
1289-   mod_dir=(instruccion & 0x80);
1290-   if (mod_dir==0)
1291-    {
1292-     dma=(instruccion & 0x7f);
1293-     DATA_MEM[dma]=valor;
1294-     bandera=2;
1295-    }
1296-   else {
1297-     DATA_MEM[REG_AUX[ARP]]=valor;
1298-     DIR_IND(instruccion);
1299-   }
1300-   despliega2(pos);
```

```
Line  Source
----- ------------------------------------------------------------------------------------------------------------------------
1301-   PC++;
1302-   CLAVE=21;
1303- }
1304-
1305- /* ------------------------------------------------------------ */
1306-
1307- OUT(instruccion)
1308- unsigned int instruccion;
1309- {
1310-   unsigned int port_dir;
1311-   port_dir=instruccion & 0x700;
1312-   port_dir=port_dir>>8;
1313-
1314-   mod_dir=(instruccion & 0x80);
1315-   if (mod_dir==0)
1316-    {
1317-     dma=(instruccion & 0x7f);
1318-     valor=DATA_MEM[dma];
1319-     bandera=2;
1320-    }
1321-   else
1322-     valor=DATA_MEM[REG_AUX[ARP]];
1323-   gotoxy(1,25);
1324-   cputs("Interrupcion provocada al Puerto :");
1325-   cprintf("%d ",port_dir);
1326-   cputs("con valor de salida : ");
1327-   cprintf("%4x",valor);
1328-   delay(6000);
1329-   PC++;
1330-   CLAVE=38;
1331- }
1332-
1333- /* ------------------------------------------------------------ */
1334-
1335- SAR(instruccion)
1336- unsigned int instruccion;
1337- {
1338-   int AR;
1339-   ARP= (instruccion&0x700);
1340-   ARP=ARP>>8;
1341-   mod_dir=(instruccion & 0x80);
1342-   if (mod_dir==0)
1343-    {
1344-     dma=(instruccion & 0x7f);
1345-     DATA_MEM[dma]=REG_AUX[ARP];
1346-     bandera=2;
1347-    }
1348-   else
1349-    {
1350-      AR=REG_AUX[ARP];
1351-      if ((instruccion & 0x20)==1)
1352-        REG_AUX[ARP]=REG_AUX[ARP]+1;
```

```
Line  Source
----- ----------------------------------------------------------------------------------------------------
1353-      if ((instruccion & 0x10)==1)
1354-        REG_AUX[ARP]=REG_AUX[ARP]-1;
1355-      DATA_MEM[AR]= REG_AUX[ARP];
1356-      if ((instruccion & 0x8)==0)
1357-        ARP=instruccion & 0x1;
1358-    }
1359-    PC++;
1360-    CLAVE=46;
1361- }
1362-
1363- /* ------------------------------------------------------------ */
1364-
1365- SACH(instruccion)
1366- unsigned int instruccion;
1367- {
1368-   unsigned int shift;
1369-   long int dato;
1370-
1371-   shift =instruccion & 0x700;
1372-   shift=shift>>8;
1373-   dato= ACC;
1374-   dato=dato<<shift;
1375-   dato=dato>>16;
1376-   mod_dir=(instruccion & 0x80);
1377-   if (mod_dir==0)
1378-    {
1379-     dma=(instruccion & 0x7f);
1380-     DATA_MEM[dma]=dato;
1381-     bandera=2;
1382-    }
1383-   else
1384-     DATA_MEM[REG_AUX[ARP]]=dato;
1385-   PC++;
1386-   CLAVE=44;
1387- }
1388-
1389- /* ------------------------------------------------------------ */
1390-
1391- SUBH(instruccion)
1392- unsigned int instruccion;
1393- {
1394-   long int dato;
1395-   mod_dir=(instruccion & 0x80);
1396-   if (mod_dir==0)
1397-    {
1398-     dma=(instruccion & 0x7f);
1399-     dato=DATA_MEM[dma];
1400-     dato=dato<<16;
1401-     ACC=ACC-dato;
1402-     bandera=2;
1403-    }
1404-   else
```

```
Line  Source
----- ----------------------------------------------------------------------------------------------------
1405-    {
1406-     dato=DATA_MEM[REG_AUX[ARP]];
1407-     dato=dato<<16;
1408-     ACC=ACC-dato;
1409-     DIR_IND(instruccion);
1410-    }
1411-    PC++;
1412-    CLAVE=52;
1413- }
1414-
1415- /* ------------------------------------------------------------ */
1416-
1417- SUBC(instruccion)
1418- unsigned int instruccion;
1419- {
1420-   long int adder_output,dato;
1421-   mod_dir=(instruccion & 0x80);
1422-   if (mod_dir==0)
1423-    {
1424-     dma=(instruccion & 0x7f);
1425-     dato=DATA_MEM[dma];
1426-     dato=dato<<16;
1427-     adder_output=ACC-dato;
1428-     if((0xffff0000 & adder_output)>= 0)
1429-        ACC=(adder_output*2)+1;
1430-      else
1431-        ACC=ACC*2;
1432-     bandera=2;
1433-    }
1434-   else
1435-    {
1436-     dato=DATA_MEM[REG_AUX[ARP]];
1437-     dato=dato<<16;
1438-     ACC=ACC-dato;
1439-     if((0xffff0000 & adder_output)>= 0)
1440-        ACC=(adder_output*2)+1;
1441-      else
1442-        ACC=ACC*2;
1443-     DIR_IND(instruccion);
1444-    }
1445-    PC++;
1446-    CLAVE=51;
1447- }
1448-
1449- /* ------------------------------------------------------------ */
1450-
1451- POP()
1452- {
1453-
1454-  ACC=sacar();
1455-  PC++;
1456-  CLAVE=40;
```

```
Line  Source
----- ----------------------------------------------------------------------------------------------------
1457-  bandera=5;
1458-
1459- }
1460-
1461- /* ------------------------------------------------------------------ */
1462-
1463- PUSH()
1464- {
1465-
1466-  meter(0xfff & ACC);
1467-  PC++;
1468-  CLAVE=41;
1469-  bandera=5;
1470-
1471- }
1472-
1473- /* ------------------------------------------------------------------ */
1474-
1475- RET()
1476- {
1477-
1478-  PC=sacar();
1479-  CLAVE=42;
1480-  bandera=5;
1481-
1482- }
1483-
1484- /* ------------------------------------------------------------------ */
1485-
1486- CALA()
1487- {
1488-
1489-  meter( (0xfff & (PC+1)) );
1490-  PC=(ACC & 0xfff);
1491-  CLAVE=16;
1492-  bandera=5;
1493-
1494- }
1495-
1496- /* ------------------------------------------------------------------ */
1497-
1498- CALL()
1499- {
1500-
1501-  meter(0xfff & (PC+2));
1502-  PC=(instruccion & 0xfff);
1503-  PC++;
1504-  CLAVE=17;
1505-  bandera=5;
1506- }
1507-
1508- /* ------------------------------------------------------------------ */
```

```
Line  Source
----- ------------------------------------------------------------------------------------------------------------------------
1509-
1510- status()
1511- {
1512-  unsigned int reg_status;
1513-
1514-  reg_status=(1 & OV);
1515-  reg_status=((reg_status<<1)|OVM);
1516-  reg_status=((reg_status<<1)|INTM);
1517-  reg_status=(reg_status<<4);
1518-  reg_status=(reg_status|0xf);
1519-  reg_status=((reg_status<<1)|ARP);
1520-  reg_status=(reg_status<<7);
1521-  reg_status=(reg_status|0x7f);
1522-  reg_status=((reg_status<<1)|DP);
1523-  return(reg_status);
1524- }
1525-
```

```
Line  Source
----- ------------------------------------------------------------------------------------------------------------------------
    - //-------------------------------------------------------------------------
    - //        Author: Alberto Garcia Castillo
    - //   Date Created: 07/07/93
    - //   Time Created: 23:33:34
    - //   Description : Conjunto de procedimientos para el manejo de ventanas y teclas
    - //                 que controlan al simulador TMS32010
    - //-------------------------------------------------------------------------
    -  #include "conio.h"
    -
   10 -  #define tam 4          /* tama$o de la pila de instrucciones */
   11 -
   12 -  #define lo(f)  ((f) & 0xff)    /* obtiene el ASCII de un cod. ext */
   13 -
   14 -  #define UP_ARROW    0x4800       /* codigos extendidos de algunas */
   15 -  #define DN_ARROW    0x5000       /* teclas para la interface */
   16 -  #define PG_UP       0x4900
   17 -  #define PG_DN       0x5100
   18 -  #define F1          0x3b00
   19 -  #define F2          0x3c00
   20 -  #define F3          0x3d00
   21 -
   22 -  struct Cola {
   23 -    unsigned int Info[tam];
   24 -    int   tope,base;
   25 -   } Stack;
   26 -
   27 -
   28 -
   29 -  int ven [7] [4] = { {1,1,80,25}, /* ventana base */
   30 -        {4,3,17,21},       /* ventana de Memoria del Prog.*/
   31 -        {54,3,77,12},      /* ventana de Registros Generales */
   32 -        {41,9,49,12},      /* ventana de la Pila */
   33 -        {54,19,76,21},     /* ventana de Registros de Estado */
   34 -        {22,3,36,21},      /* ventana de Memoria de Datos */
   35 -        {41,15,77,16} };   /* ventana con la instrucci"nactual*/
   36 -
   37 -  signed int DATA_MEM[144];
   38 -  unsigned int MEM_PROG[100] ;
   39 -  int w,v,mod,r_m,bandera=0,
   40 -      reg,EA,ext,li,ls,
   41 -      num_op,override,
   42 -      flg_hlt,auxb,pos;
   43 -
   44 - /*-----------------------------------------------------------------*/
   45 -
   46 -  void marco (int uplx,int uply,int lorx,int lory,char *s)
   47 -
   48 -  {
   49 -    int i,xs;
   50 -
   51 -    gotoxy(uplx,uply);
   52 -    putch(201);
```

```
ine  Source
---- ------------------------------------------------------------------------------------------------------------------------
3  -    for (i=(uplx+1);i <= (lorx-1);i++) putch(205);
4  -    putch(187);
5  -    for (i=(uply+1);i <= (lory-1);i++) {
6  -      gotoxy(uplx,i); putch(186);
7  -      gotoxy(lorx,i); putch(186);
8  -      }
9  -    gotoxy(uplx,lory);
0  -    putch(200);
1  -    for (i=(uplx+1);i <= (lorx-1);i++) putch(205);
2  -    putch(188);
3  -    if (*s != 0) {
4  -      xs = ((lorx-uplx-strlen(s)+1)/2) + uplx;
5  -      if (xs < uplx) xs = uplx;
6  -      for (i=0;(*s != 0) && (i < (lorx-uplx+1));i++) {
7  -        gotoxy(xs+i,uply); putch(*s++);
8  -        }
9  -      }
0  -  }
1  -
2  -
3  - /* ------------------------------------------------------------------ */
4  -
5  -  char lee_tecla (int *k)
6  -
7  -  {
8  -    union REGS regs;
9  -
0  -    regs.h.ah = 0;
1  -    int86(0x16,&regs,&regs);
2  -    *k = regs.x.ax;
3  -    if (lo(*k) == 3) exit(1);
4  -    return regs.h.al;
5  -  }
6  -
7  - /* ------------------------------------------------------------------ */
8  -
9  -  void beep (void)
0  -
1  -  {
2  -    sound(750);
3  -    delay(200);
4  -    nosound();
5  -  }
6  -
7  -
8  - /* ------------------------------------------------------------*/
9  - /*   Dibuja las ventanas del simulador TMS32010 en Video      */
00 -
01 -
02 -  void pantalla ()
03 -
04 -  {
```

```
ine  Source
---- ------------------------------------------------------------------------------------------------------------------------
05 -  clrscr();
06 -  gotoxy(25,1);
07 -  cprintf(">> Simulador TMS32010 <<");
08 -  marco(ven[1] [0]-1,ven[1] [1]-1,ven[1] [2]+1,ven[1] [3]+1," Mem. Prog.");
09 -  marco(ven[2] [0]-1,ven[2] [1]-1,ven[2] [2]+1,ven[2] [3]+1," Regs. Gerals.");
10 -  marco(ven[3] [0]-1,ven[3] [1]-1,ven[3] [2]+1,ven[3] [3]+1," Pila ");
11 -  marco(ven[4] [0]-1,ven[4] [1]-1,ven[4] [2]+1,ven[4] [3]+1," Regs.de Edo.");
12 -  marco(ven[5] [0]-1,ven[5] [1]-1,ven[5] [2]+1,ven[5] [3]+1," Mem. de Datos ");
13 -  marco(ven[6] [0]-1,ven[6] [1]-1,ven[6] [2]+1,ven[6] [3]+1," Instrucci'n ");
14 -  }
15 -
16 - /* ------------------------------------------------------------------------ */
17 -
18 -  void pantalla2()
19 - {
20 -
21 - marco(ven[6] [0]-1,ven[6] [1]-1,ven[6] [2]+1,ven[6] [3]+1," Memoria de Datos ");
22 -
23 - }
24 -
25 - /* ------------------------------------------------------------------------*/
26 -
27 -  void sel_ventana (int win)
28 -
29 -  {
30 -    window(ven[win] [0],ven[win] [1],ven[win] [2],ven[win] [3]);
31 -  }
32 -
33 -
34 - /* ------------------------------------------------------------------------*/
35 - /*     Actualiza la ventana de que fue alterada por la operacion          */
36 -
37 -
38 -  void actz_ventana (int win,int dato,unsigned char scroll)
39 -
40 -  {
41 -    lowvideo();
42 -    sel_ventana(win);
43 -    gotoxy(2,1);
44 -    if (scroll) {
45 -      delline();
46 -      gotoxy(2,ven[win] [3] - ven[win] [1]+1);
47 -      }
48 -    else insline();
49 -    cprintf("%4x [ %4x",dato,MEM_PROG[dato]);
50 -    normvideo();
51 -    sel_ventana(0);
52 -  }
53 -
54 - /* ------------------------------------------------------------------------*/
55 - /*             Despliega la ventana de Memoria de Datos                   */
56 -
```

```
ine  Source
---- ------------------------------------------------------------------------------------------------------------------------
57 -
58 -  void despliega2(int pos)
59 -  {
60 -   int x;
61 -
62 -   sel_ventana(5);
63 -   for (x=0;x(= 18;x++) {
64 -      gotoxy(2,x+1);
65 -      cprintf("%4x [ %4x",pos+x,DATA_MEM[pos+x]);
66 -      }
67 -    sel_ventana(0);
68 -  }
69 -
70 - /* ---------------------------------------------------------------------*/
71 - /*          Despliega la ventana de Memoria del Programa                */
72 -
73 -
74 - void despliega (int loc)
75 -
76 -  {
77 -    int x;
78 -
79 -    sel_ventana(1);
80 -    for (x=0;x(= 18;x++) {
81 -      gotoxy(2,x+1);
82 -      cprintf("%4x [ %4x",loc+x,MEM_PROG[loc+x]);
83 -      }
84 -    li = loc; ls = loc+x-1;
85 -    sel_ventana(0);
86 -  }
87 -
88 -
89 - /* ---------------------------------------------------------------------*/
90 - /*          Despliega la ventana de registros generales                 */
91 -
92 -
93 - void registros ()
94 -
95 -  {
96 -
97 -    sel_ventana(2);
98 -    gotoxy(3,2);  cputs("Id.");
99 -    gotoxy(3,4);  cprintf(" 1 - ACC         ");
00 -    gotoxy(3,4);  cprintf(" 1 - ACC %1x",ACC);
01 -    gotoxy(3,5);  cprintf(" 2 - P           ");
02 -    gotoxy(3,5);  cprintf(" 2 - P   %1x",P);
03 -    gotoxy(3,6);  cprintf(" 3 - AR0         ");
04 -    gotoxy(3,6);  cprintf(" 3 - AR0 %1x",REG_AUX[0]);
05 -    gotoxy(3,7);  cprintf(" 4 - AR1         ");
06 -    gotoxy(3,7);  cprintf(" 4 - AR1 %1x",REG_AUX[1]);
07 -    gotoxy(3,8);  cprintf(" 5 - T         ");
08 -    gotoxy(3,8);  cprintf(" 5 - T   %4x",T);
```

```
ine  Source
---- ------------------------------------------------------------------------------------------------------------------------
09 -    gotoxy(3,9); cprintf(" 6 -          ");
10 -    gotoxy(3,9); cprintf(" 6 - PC   %3x",PC);
11 -    gotoxy(3,10); cprintf(" 7 -          ");
12 -    gotoxy(3,10); cprintf(" 7 - BIO   %1d",BIO);
13 -    sel_ventana(0);
14 -  }
15 -
16 - /* ------------------------------------------------------------------ */
17 -
18 - void ver_inst()
19 - {
20 -  char modo;
21 -
22 -  sel_ventana(6);
23 -
24 -    if (mod_dir==0)
25 -       modo='D';
26 -    else
27 -       modo='I';
28 -
29 -    gotoxy(2,1);  cprintf("                                  ");
30 -    switch(bandera)
31 -      {
32 -        case 0:
33 -        case 1: gotoxy(2,1);  cprintf(" INS.           ");
34 -         gotoxy(13,1); cprintf(" dma       ");
35 -         gotoxy(25,1); cprintf(" MOD.      ");
36 -         break;
37 -        case 2: gotoxy(2,1);  cprintf(" INS.           ");
38 -         gotoxy(2,1);  cprintf(" INS. %s",INSTR[CLAVE]);
39 -         gotoxy(13,1); cprintf(" dma  %4x",dma);
40 -         gotoxy(25,1); cprintf(" MOD. %c",modo);
41 -         break;
42 -        case 3: gotoxy(2,1);  cprintf(" INS.           ");
43 -         gotoxy(2,1);  cprintf(" INS. %s",INSTR[CLAVE]);
44 -         gotoxy(13,1); cprintf(" MOD. %c",modo);
45 -         break;
46 -        case 4: gotoxy(2,1);  cprintf(" INS.           ");
47 -         gotoxy(2,1);  cprintf(" INS. %s",INSTR[CLAVE]);
48 -         gotoxy(13,1); cprintf(" CTE. %1x",ACC);
49 -         break;
50 -        case 5: gotoxy(2,1);  cprintf(" INS.           ");
51 -         gotoxy(2,1);  cprintf(" INS. %s",INSTR[CLAVE]);
52 -         break;
53 -        case 6: gotoxy(2,1);  cprintf(" INS.           ");
54 -         gotoxy(2,1);  cprintf(" INS. %s",INSTR[CLAVE]);
55 -         gotoxy(13,1); cprintf(" CTE. %4x",REG_AUX[arp_aux]);
56 -         break;
57 -        case 7: gotoxy(2,1);  cprintf(" INS.           ");
58 -         gotoxy(2,1);  cprintf(" INS. %s",INSTR[CLAVE]);
59 -         gotoxy(13,1); cprintf(" CTE. %1u",cte1);
60 -         break;
```

```
Line  Source
----- ----------------------------------------------------------------------------------------------------
161 -      }
162 -    sel_ventana(0);
163 -
164 - }
165 -
166 -
167 - /* ------------------------------------------------------------------------ */
168 -
169 - void Programa(int loc)
170 -  {
171 -   int loc_a;
172 -
173 -   sel_ventana(5);
174 -
175 -   loc_a=loc;
176 -   gotoxy(3,2);  cprintf("%4x",DATA_MEM[loc]);loc++;
177 -   gotoxy(9,2);  cprintf("%4x",DATA_MEM[loc]);loc++;
178 -   gotoxy(15,2); cprintf("%4x",DATA_MEM[loc]);loc++;
179 -   gotoxy(21,2); cprintf("%4x",DATA_MEM[loc]);loc++;
180 -   gotoxy(27,2); cprintf("%4x",DATA_MEM[loc]);loc++;
181 -   gotoxy(33,2); cprintf("%4x",DATA_MEM[loc]);loc++;
182 -   gotoxy(39,2); cprintf("%4x",DATA_MEM[loc]);loc++;
183 -
184 -   gotoxy(3,3);  cprintf("%4x",loc_a);loc_a++;
185 -   gotoxy(9,3);  cprintf("%4x",loc_a);loc_a++;
186 -   gotoxy(15,3); cprintf("%4x",loc_a);loc_a++;
187 -   gotoxy(21,3); cprintf("%4x",loc_a);loc_a++;
188 -   gotoxy(27,3); cprintf("%4x",loc_a);loc_a++;
189 -   gotoxy(33,3); cprintf("%4x",loc_a);loc_a++;
190 -   gotoxy(39,3); cprintf("%4x",loc_a);loc_a++;
191 -   sel_ventana(0);
192 -
193 - }
194 -
195 - /* ------------------------------------------------------------------------*/
196 - /*           Despliega en pantalla el contenido de la Pilas               */
197 -
198 -
199 -  void Pila ()
200 -
201 -  {
202 -    int k,tope;
203 -
204 -    sel_ventana(3);
205 -
206 -    for (k=0;k<=3;k++) {
207 -      gotoxy(4,k+1);
208 -      cprintf("%3x",Stack.Info[(k%4)]);
209 -      }
210 -    sel_ventana(0);
211 -  }
212 -
```

```
Line  Source
---- ------------------------------------------------------------------------------------------------------------------------
313 - /* ------------------------------------------------------------------ */
314 - /*         Despliega la ventana con los Registros de estado           */
315 -
316 -
317 -  void banderas ()
318 -
319 -  {
320 -    sel_ventana(4);
321 -    gotoxy(2,1);  cputs("OV OVM INTM ARP DP");
322 -    gotoxy(2,2);  cprintf("%1d",OV);
323 -    gotoxy(6,2);  cprintf("%1d",OVM);
324 -    gotoxy(10,2); cprintf("%1d",INTM);
325 -    gotoxy(14,2); cprintf("%1d",ARP);
326 -    gotoxy(19,2); cprintf("%1d",DP);
327 -    sel_ventana(0);
328 -  }
329 -
330 -
331 - /* ------------------------------------------------------------------*/
332 -
333 - void loc_mem (int *loc,int cual)
334 -
335 -  {
336 -    gotoxy(1,25);
337 -    cputs("Localidad de Memoria : ");
338 -    cscanf("%4x",loc);
339 -    if (cual == 1)
340 -    {
341 -    if ((*loc >= 0) && (*loc <= 143)) {     /* MEMORIA DE DATOS */
342 -      if (*loc <= 9) li = 0;
343 -      else if (*loc >= 144) li = 125;
344 -      else li = *loc - 9;
345 -      despliega2(li);
346 -      }
347 -    }
348 -    else {                                 /* MEMORIA DE PROGRAMA */
349 -     if ((*loc >= 0) && (*loc <= 1023)) {
350 -      if (*loc <= 9) li = 0;
351 -      else if (*loc >= 1024) li = 1017;
352 -      else li = *loc - 9;
353 -      despliega(li);
354 -      }
355 -     }
356 -  }
357 -
358 - /* ------------------------------------------------------------------*/
359 -
360 -  void mod_mem ()
361 -
362 -  {
363 -    int cas;
364 -
```

```
Line  Source
---- ------------------------------------------------------------------------------------------------------------------------
365 -    auxb=0;
366 -    gotoxy(1,25);
367 -    cputs("Localidad de Memoria : ");
368 -    cscanf("%4x",&cas);
369 -    if ((cas >= 0) && (cas <= 143)) {     /* MEMORIA DE DATOS */
370 -      if (cas <= 9) li = 0;
371 -      else if (cas >= 144) li = 125;
372 -      else li = cas - 9;
373 -      despliega2(li);
374 -      }
375 -    gotoxy(29,25); cputs("contenido : ");
376 -    cscanf("%4x",&auxb);
377 -    DATA_MEM[cas] = auxb;
378 -    despliega2(li);
379 -  }
380 -
381 -
382 - /* ---------------------------------------------------------------------*/
383 -
384 -   void mod_prog()
385 -   {
386 -   int casilla;
387 -
388 -    auxb=0;
389 -    gotoxy(1,25);
390 -    cputs("Localidad de Memoria : ");
391 -    cscanf("%4x",&casilla);
392 -    if ((casilla >= 0) && (casilla <= 1023)) {
393 -      if (casilla <= 9) li = 0;
394 -      else if (casilla >= 1024) li = 1017;
395 -      else li = casilla - 9;
396 -      despliega(li);
397 -      }
398 -    gotoxy(29,25); cputs("contenido :");
399 -    cscanf("%4x",&auxb);
400 -    MEM_PROG[casilla] =auxb;
401 -    despliega(li);
402 -
403 -   }
404 -
405 - /* ---------------------------------------------------------------------*/
406 - /*        Modifica alguno de los registros de estado                    */
407 -
408 -
409 -   void mod_edo()
410 -   {
411 -
412 -   char ban;
413 -
414 -   gotoxy(1,25);
415 -   cputs("Escriba las iniciales del Registro O/M/I/A/D a modificar :");
416 -   cscanf("%1c",&ban);
```

```
ine  Source
---- ------------------------------------------------------------------------------------------------------------------------
117 -   switch(toupper(ban)) {
118 -   case 'O'  : if (OV==1)       /* ESTADO = status(); */
119 -    OV=0;
120 -        else
121 -    OV=1;
122 -        break;
123 -   case 'M'  : if (OVM == 1)
124 -    OVM=0;
125 -         else
126 -     OVM=1;
127 -         break;
128 -   case 'I'  :  if (INTM == 1)
129 -    INTM=0;
130 -         else
131 -    INTM=1;
132 -         break;
133 -   case 'A'  :  if (ARP == 1)
134 -    ARP=0;
135 -         else
136 -    ARP=1;
137 -         break;
138 -   case 'D'  :  if (DP == 1)
139 -    DP=0;
140 -         else
141 -    DP=1;
142 -         break;
143 -    }
144 -   ESTADO=status();
145 -   banderas();
146 -   }
147 -
148 - /* ------------------------------------------------------------------------*/
149 - /*            Modifica alguno de los Registros Generales                   */
150 -
151 -
152 -  void mod_reg ()
153 -
154 -  {
155 -    int id,val;
156 -    long int val2;
157 -
158 -    gotoxy(1,25); cputs("Id. del registro : ");
159 -    cscanf("%2d",&id);
160 -    gotoxy(23,25); cputs("valor : ");
161 -    if (id ==1 || id ==2)
162 -       cscanf("%lx",&val2);
163 -    else
164 -       cscanf("%4x",&val);
165 -    if (id >= 1 && id <= 8) {
166 -      switch(id) {
167 -        case 1  : ACC = val2; break;
168 -        case 2  : P   = val2; break;
```

```
Line  Source
----- ----------------------------------------------------------------------------------------------------
469 -        case 3  : REG_AUX[0] = val; break;
470 -        case 4  : REG_AUX[1] = val; break;
471 -        case 5  : T   = val; break;
472 -        case 6  : PC  = val;
473 -    despliega(PC);
474 -    Pila();
475 -    break;
476 -        case 7  : BIO  = val;
477 -    break;
478 -
479 -        }
480 -      registros();
481 -      }
482 -  }
483 -
484 - /* ------------------------------------------------------------------*/
485 -
486 - void mod_inst()
487 -
488 -  {
489 -    int  val;
490 -    char m_dir,inst_id;
491 -
492 -    gotoxy(1,25); cputs("Cambiar (I)nst. (D)ma ' (M)odo : ");
493 -    cscanf("%c",&inst_id);
494 -    gotoxy(38,25); cputs("valor : ");
495 -    if (inst_id=='M' || inst_id== 'I')
496 -       cscanf("%4x",&val);
497 -    else
498 -       cscanf("%c",&m_dir);
499 -    switch(toupper(inst_id)) {
500 -        case 'M': mod_dir = val; break;
501 -        case 'I': INS = val;     break;
502 -        case 'D': dma = m_dir;   break;
503 -      }
504 -    despliega(PC);
505 -    ver_inst();
506 -  }
507 -
508 -
509 - /* ------------------------------------------------------------------ */
510 - /*              Incerta elemento en el tope de la Pila                */
511 -
512 -   void meter(unsigned int entrada)
513 -   {
514 -
515 -    if ((Stack.tope == 0) && (Stack.Info[Stack.tope] !='\0')) {
516 -       Stack.tope=3;
517 -       for (i=0;i<=2;++i)
518 -          Stack.Info[i]=Stack.Info[i+1];
519 -      }
520 -    Stack.Info[Stack.tope] = entrada;
```

```
Line  Source
----- ----------------------------------------------------------------------------------------------------
521 -    Stack.tope =((++Stack.tope)%4);
522 -
523 -   }
524 -
525 - /*-------------------------------------------------------------------*/
526 - /*            Elimina el elemento del tope de la Pila                */
527 -
528 -   unsigned int sacar()
529 -   {
530 -
531 -    if ((Stack.Info[Stack.tope] !='\0') && (Stack.tope ==0)) {
532 -       Stack.tope=3;
533 -       salida=Stack.Info[Stack.tope];
534 -       Stack.Info[Stack.tope]='\0';
535 -       }
536 -    else {
537 -         if (Stack.tope !=0) {
538 -            Stack.tope=((--Stack.tope)%4);
539 -            salida=Stack.Info[Stack.tope];
540 -     Stack.Info[Stack.tope]='\0';
541 -          }
542 -       }
543 -    return(salida);
544 -
545 -   }
546 -
547 - /*-------------------------------------------------------------------*/
548 - /*  Controla todos los movimientos en todas las ventanas del Simulador  */
549 -
550 -
551 -  void interface ()
552 -
553 -  {
554 -    int k,pos;
555 -    int fl = 1;
556 -
557 -    despliega(PC);
558 -    despliega2(0);
559 -    registros();
560 -    Pila();
561 -    banderas();
562 -    ver_inst();
563 -    pos=0;
564 -    while (fl) {
565 -      gotoxy(1,24);
566 -      cputs("F3.Inst./(R)egs.Sen./F2-Regs.Edo./(M)em.Prog./(L)oc.Datos/(C)ont. PgUp PgDn ");
567 -      putch(24);putch(25);
568 -      gotoxy(1,25);
569 -      clreol();
570 -      cputs("Comando : ");
571 -      lee_tecla(&k);
572 -      switch (k) {
```

```
Line  Source
----- ----------------------------------------------------------------------------------------------------
573 -       case  UP_ARROW : if (li > 0) {
574 -                          actz_ventana(1,--li,0);
575 -                          --ls;
576 -                          }
577 -                        break;
578 -       case  DN_ARROW : if (ls < 1023) {
579 -                          actz_ventana(1,++ls,1);
580 -                          ++li;
581 -                          }
582 -                        break;
583 -       case  PG_UP    : if (li > 0) {
584 -                          if (li-19 >= 0) li = li-19;
585 -                          else li = 0;
586 -                          despliega(li);
587 -                          }
588 -                        break;
589 -       case  PG_DN    : if (ls < 1023) {
590 -                     if (ls+19 <= 1023) li = ls+1;
591 -                     else li = ls - (ls+19-1024);
592 -                             despliega(li);
593 -                          }
594 -                        break;
595 -       case  0x1352   :                              /* Registros */
596 -       case  0x1372   : gotoxy(1,24); clreol();
597 -                        cputs("Modificacion de Registros");
598 -                        mod_reg();
599 -                        break;
600 -       case  0x324d   :                              /*  Memoria  */
601 -       case  0x326d   : gotoxy(1,24); clreol();
602 -                        cputs("Modificacion de Memoria");
603 -                  mod_prog();
604 -                        break;
605 -       case  0x264c   :                             /*  Localizar */
606 -       case  0x266c   : gotoxy(1,24); clreol();
607 -                      cputs("Localizar casilla de Memoria");
608 -                  mod_mem();
609 -                  break;               /* Registros de Estado */
610 -       case  F2       : gotoxy(1,24); clreol();
611 -                  cputs("Modifica los Registros de estado");
612 -                  mod_edo();
613 -                  break;
614 -       case  F3       : gotoxy(1,24); clreol();
615 -                  cputs("Modifica Instrucci"n ");
616 -                  mod_inst();
617 -                  break;
618 -       case  0x2e43   :                              /*  Continuar */
619 -       case  0x2e63   : fl = 0; break;
620 -       case  0x011b   : clrscr();
621 -                  printf("Salida del Simulador TMS32010.");
622 -                  exit(0);
623 -       default        : beep();
624 -     }
```

```
ine  Source
---- ------------------------------------------------------------------------------------------------------------------------
25 -    }
26 -
27 -  }
28 -
```

```
Line  Source
----- ----------------------------------------------------------------------------------------------------------------
   1  - unsigned int  PC=0, OV, BIO, OVM, INTM, DP, ARP, TOS, REG_AUX[1],entrada,salida;
   2  - unsigned int MEM_PROG[100], ESTADO, mod_dir, CONT,instruccion,INS,CLAVE,dma;
   3  - unsigned int  arp_aux, cte1;
   4  - signed int  T, DATA_MEM[144];
   5  - long int ACC, P;
   6  - char INSTR[60][5] = {"ABS","ADD","ADDH","ADDS","AND","APAC","B","BANZ","BGEZ",
   7  -   "BGZ","BIOZ","BLEZ","BLZ","BNZ","BV","BZ","CALA","CALL",
   8  -   "DINT","DMOV","EINT","IN","LAC","LACK","LAR","LARK","LARP",
   9  -   "LDP","LDPK","LST","LT","LTA","LTD","MAR","MPY","MPYK","NOP",
  10  -   "OR","OUT","PAC","POP","PUSH","RET","ROVM","SACH","SACL",
  11  -   "SAR","SOVM","SPAC","SST","SUB","SUBC","SUBH","SUBS","TBLR",
  12  -   "TBLW","XOR","ZAC","ZALH","ZALS"};
  13  - int valor,i;
```

```
Line  Source
----- ----------------------------------------------------------------------------------------------------
   - #include "math.h"
   - #include "conio.h"
   - #include "stdarg.h"
   - #include "graphics.h"
   -
   - int    GraphDriver;
   - int    GraphMode;
   - int    MaxX, MaxY;
   - int    MaxColors;
   - int    ErrorCode;
   -
   - void Inicia(void);
   - void Pant(void);
   - void Presentacion();
   - void StatusLine(char *msg);
   - void DibBorde(void);
   - void changetextstyle(int font, int direction, int charsize);
   -
   - comenzar()
   - {
   -   Inicia();
   -   Pant();
   -   closegraph();
   -   return(0);
   - }
   -
   - /* ------------------------------------------------------------------ */
   - void Inicia(void)
   - {
   -   GraphDriver = DETECT;
   -   initgraph( &GraphDriver, &GraphMode, "" );
   -   ErrorCode = graphresult();
   -   if( ErrorCode != grOk ){
   -     printf(" Graphics System Error: %s\n", grapherrormsg( ErrorCode ) );
   -     exit( 1 );
   -   }
   -   MaxColors = getmaxcolor() + 1;
   -   MaxX = getmaxx();
   -   MaxY = getmaxy();
   - }
   -
   - /* ------------------------------------------------------------------ */
   - void Pant(void)
   - {
   -   struct viewporttype info;
   -   int h, w;
   -   Presentacion();
   -   getviewsettings( &info );
   -   changetextstyle( GOTHIC_FONT, HORIZ_DIR, 4 );
   -   settextjustify( CENTER_TEXT, CENTER_TEXT );
   -   h = info.bottom - info.top;
   -   w = info.right  - info.left;
```

```
Line  Source
----- ----------------------------------------------------------------------------------------------------------------
 53 -   outtextxy( w/2, h/2.5, "SIMULADOR TMS32010" );
 54 -   changetextstyle( DEFAULT_FONT, HORIZ_DIR, 0.5 );
 55 -   settextjustify( CENTER_TEXT, CENTER_TEXT );
 56 -   outtextxy( w/2, (h/2)*1.1, "REALIZADO POR :" );
 57 -   outtextxy( w/2, (h/2)*1.2, "MA. DEL PILAR VILLA CERVANTES" );
 58 -   outtextxy( w/2, (h/2)*1.3, "ALBERTO GARCIA CASTILLO" );
 59 -   outtextxy( w/2, (h/2)*1.5, "MAYO-93" );
 60 -   StatusLine( "PRESIONA CUALQUIER TECLA PARA CONTINUAR" );
 61 -   getch();
 62 -   cleardevice();
 63 - }
 64 -
 65 - /* ------------------------------------------------------------------ */
 66 - void Presentacion()
 67 - {
 68 -   int height;
 69 -   cleardevice();
 70 -   setcolor( MaxColors - 1 );
 71 -   setviewport( 0, 0, MaxX, MaxY, 1 );
 72 -   height = textheight( "H" );
 73 -   changetextstyle( DEFAULT_FONT, HORIZ_DIR, 1 );
 74 -   settextjustify( CENTER_TEXT, TOP_TEXT );
 75 -   setviewport( 0, height+4, MaxX, MaxY-(height+4), 1 );
 76 -   DibBorde();
 77 -   setviewport( 1, height+5, MaxX-1, MaxY-(height+5), 1 );
 78 - }
 79 -
 80 - /* ------------------------------------------------------------------ */
 81 - void StatusLine( char *msg )
 82 - {
 83 -   int height;
 84 -   setviewport( 0, 0, MaxX, MaxY, 1 );
 85 -   setcolor( MaxColors - 1 );
 86 -   changetextstyle( DEFAULT_FONT, HORIZ_DIR, 1 );
 87 -   settextjustify( CENTER_TEXT, TOP_TEXT );
 88 -   setlinestyle( SOLID_LINE, 0, NORM_WIDTH );
 89 -   setfillstyle( EMPTY_FILL, 0 );
 90 -   height = textheight( "H" );
 91 -   bar( 0, MaxY-(height+4), MaxX, MaxY );
 92 -   rectangle( 0, MaxY-(height+4), MaxX, MaxY );
 93 -   outtextxy( MaxX/2, MaxY-(height+2), msg );
 94 -   setviewport( 1, height+5, MaxX-1, MaxY-(height+5), 1 );
 95 -
 96 - }
 97 -
 98 - /* ------------------------------------------------------------------ */
 99 - void DibBorde(void)
100 - {
101 -   struct viewporttype vp;
102 -   setcolor( MaxColors - 1 );
103 -   setlinestyle( SOLID_LINE, 0, NORM_WIDTH );
104 -   getviewsettings( &vp );
```

```
ine  Source
---- ------------------------------------------------------------------------------------------------------------
05 -   rectangle( 0, 0, vp.right-vp.left, vp.bottom-vp.top );
06 - }
07 -
08 - /* ------------------------------------------------------------------ */
09 -
10 - void changetextstyle(int font, int direction, int charsize)
11 - {
12 -   int ErrorCode;
13 -
14 -   graphresult();
15 -   settextstyle(font, direction, charsize);
16 -   ErrorCode = graphresult();
17 -   if( ErrorCode != grOk ){
18 -     closegraph();
19 -     printf(" Error Grafico: %s\n", grapherrormsg( ErrorCode ) );
20 -     exit( 1 );
21 -   }
22 - }
```